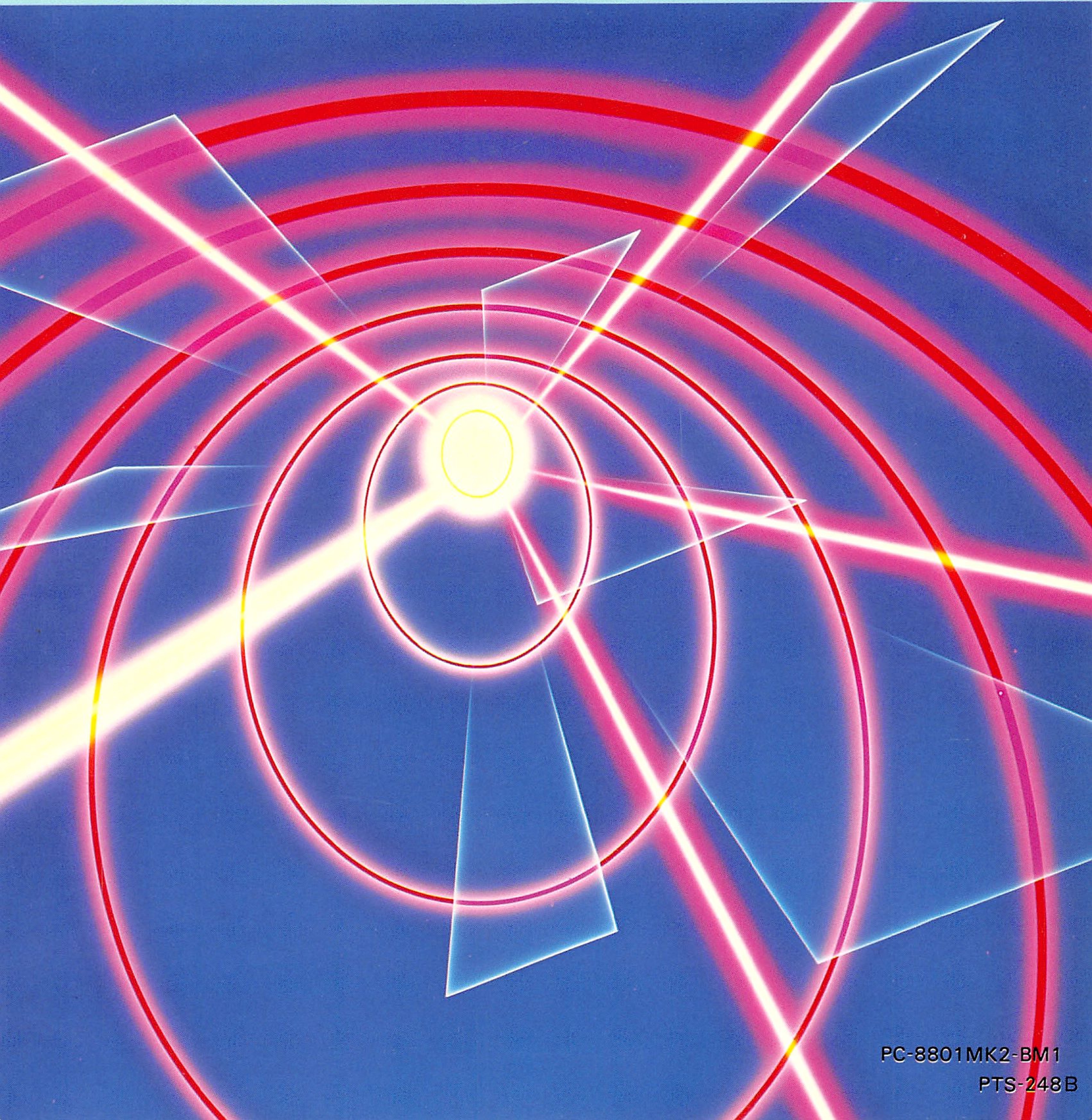
NEC Personal Computer

PC-8801mkII
BASIC
REFERENCE MANUAL
N_{88}-BASIC・N-BASIC



PC-8801MK2-BM1
PTS-248B

BASIC

REFERENCE MANUAL

N_{88}-BASIC・N-BASIC

まえがき

パーソナルコンピュータ PC-8801MKIIは，プログラミング言語として，N_{88}-BASICとN-BASICの２つのBASICを搭載しています．

本書はこれらのBASICの命令を個々について，詳細に解説したリファレンスマニュアルです．このマニュアルを読む前に，PC-8801MKII ユーザーズマニュアルをよく読んで，基本的な操作方法を習得することをお勧めします．

このマニュアルの構成

本書は各機能，命令などを分かりやすくするため，1，2，3，4章の独立した章とAPPENDIXという構成になっています．

またN_{88}- BASICとN - BASICの解説は分離せず一括して行います．ただし，N_{88}- BASICの方がN - BASICの上位互換性を持っていますから，原則としてN_{88}- BASICを題材に解説し，特に内容の異なるものや片方にしかない機能の解説の場合はそのつどN_{88}- BASIC，N - BASICと明記し注意書きを添えてあります．

またサンプルプログラムについては，N - BASIC専用命令の解説を除いて，すべてN_{88}- BASICを題材として作られています．実際に試してみる場合はN_{88}-BASICのモードで実行してください．

1章　N_{88}- BASIC（N - BASIC）の概要

N_{88}-BASIC（N-BASIC）の特徴や機能をコマンド・ステートメントなどの説明に先だって解説しています．その内容は，BASIC言語の定数，変数の意味，演算の決まりなど，BASIC言語を使う上での最低限の知識と，N_{88}-BASICに新しく追加された機能の紹介や，それを使う上での知識です．これらは2章，3章，4章の文法の説明を理解する上でどうしても必要な事項です．必ず読むようにしてください．

2章　コマンド・ステートメント

3章　関　数

BASICの機能は大きく分けてコマンド・ステートメントと関数に分類することができます．

コマンド及びステートメントはBASICが持つ命令で，コマンドは一般にダイレクトモードで使用するもので実行後〝Ok〞が表示されます．ステートメントは主にプログラム中で使用します（明確に分かれている訳ではない）．

関数は何らかの命令に付随して引用される機能です．

4章　拡張命令

N_{88}-BASICには拡張命令パッケージを付加することができます．

この章では，追加されるステートメントについて説明します．

APPENDIX

ここでは，2章，3章，4章で解説したN_{88}-BASICの個々の命令が実際のプログラミングではどう使われるかを，サンプルプログラムを中心に説明します．

またその他に，各周辺機器との入出力の仕方，ファイルの作り方，機械語プログラムの作り方や各種の一覧表などもここでまとめて取り扱っています．

目　次

第2章　コマンド・ステートメント

第3章　関　　数

第4章　拡張命令

APPENDIX

機能別索引

コマンド

一般命令

(ステートメント)

画面・グラフィック命令

●画面制御命令

●キャラクタ座標系及びテキスト画面に対する命令

●ワールド座標系に対する命令

●N-BASIC専用命令

（ステートメント）

（関数）

算術命令

（関数）

文字列操作命令

（ステートメント）

（関数）

入出力命令

（ステートメント）

キー制御命令

（ステートメント）

クロック制御命令

（ステートメント）

（関数）

プリンタ制御命令

(ステートメント)

(関数)

ライトペン制御命令

(ステートメント)

(関数)

RS-232C制御命令

(ステートメント)

●N-BASIC専用命令

(ステートメント)

（関数）

ディスク制御命令

（ステートメント）

（関数）

特殊命令

（ステートメント）

拡張命令

（ステートメント）

第1章

N88-BASIC（N-BASIC）の概要

1.1　N_{88}-BASIC と N-BASIC

PC-8801MKIIは，N_{88}-BASICとN-BASICという2つのBASICを持っており，必要に応じて切り替えて使用します．

N-BASICは，姉妹機のPC-8001にに搭載されているものと全く同じで，現在数多く流通しているPC-8001用のソフトウェアを利用して頂くために用意されています．ただし，このモードでは，PC-8801MKII が持っている多くの重宝なハードウェア（高分解能グラフィックスなど）を完全にサポートすることはできません．

N_{88}-BASICは，PC-8801MKIIのメインプログラミング言語で，PC-8801と完全に互換性があります．さらに，拡張命令を追加することが可能となっています．

この2つのBASICは原則的にコンパチビリティを持っています．ただし，ハードウェアが変ったことやBASICが改良されたことにより，なくなった機能や内容の変ったものがいくつかあります．ここでN_{88}- BASICとN - BASICの基本的な違いについて説明しておきます．

次に示すのは，N - BASICにはあって，N_{88}- BASICではなくなった命令と機能です．

MOUNT 命令（ディスク）
REMOVE 命令（ディスク）
FORMAT 命令（ディスク）
CLOAD／CLOAD？命令（カセット）
CSAVE 命令（カセット）
アトリビュート方式のセミグラフィック機能
キャラクタによって線を引く機能

また，一命令という意味ではなく，BASICの内部的な処理でその内容が変ったものがいくつかあります．

1つは変数名における制限です．N_{88}- BASICでは40文字までの名前を完全に区別できますが，N - BASICでは最初の2文字までしか判別の対象になりません．

もう1つは内部の演算処理で実数値を整数値に変換する方法が異なることです．N_{88}- BASICは小数点以下を四捨五入して整数化しますが，N - BASICでは小数点以下を切り捨てます．

またエラーメッセージの出力の仕方も異なります．N-BASICおよびN_{88}DISK

-BASICではフルセンテンスのメッセージを出力し，N_{88}-BASIC（ROMバージョン）では簡略された形で出力されます。詳しくはAPPENDIXのエラーコード表を参照ください．

1.2 N_{88}-BASICの主な特徴

N_{88}-BASICは新しい機能として次のような特徴を持っています．

1. PC-8801MKIIが持っているハードウェアの機能のほとんどをN_{88}-BASICで使用可能です．
   操作可能な機器を以下に示します．
   ○プログラマブル・ファンクションキー
   ○オーディオカセット
   ○ RS-232Cインターフェイス機器
   ○リアルタイムタイマ
   ○ライトペン
   ○ミニフロッピーディスク
   ○標準フロッピーディスク
   ○セントロニクス・インターフェイス機器（プリンタ）
2. 48Kバイトものグラフィックス専用RAMを持っているため，640×200（8色カラー1ページまたは白黒3ページ），640×400（白黒1ページ）の高分解能グラフィックスを実現できます．
3. またこれをサポートする種々のグラフィック命令は強力で，特にWINDOW，VIEWによる論理的な座標系の設定は，プログラミングを簡略にし，表示を多彩にします．
4. リストなどキャラクタを表示するテキストRAMと，グラフィックRAMは完全に区別されているため，テキスト画面とグラフィック画面の合成が簡単にできます．
5. パレットという新しい概念を導入したことにより，多彩なカラー表示が可能です．
6. プログラム中の飛び先の指定としてラベル名が使えるため，プログラム開発，デバッグが飛躍的に向上します．
7. 各ハードウェアからの割り込みをBASIC中で操作できます．
8. 漢字ROMが標準装備されており，日本語の文章を処理することができます．
9. IF　THEN　ELSE／WHILE　WEND（ディスクバージョン）等の文によ

り構造化されたプログラムにすることができます．

10．変数名は，40字まで使用可能であるため，分かりやすい変数名を付けることができます．

11. N_{88}DISK-BASIC を導入の際は，倍精度関数が準備されており，科学技術計算にも適応できます．また大規模なプログラムに対して，オーバーレイ機能も用意されており，必要な変数の受け渡しも可能です．

12. 汎用 I/O ポートを持っているためバーコードリーダなどが接続できます．

13. 拡張命令パッケージを付加することにより，タートルグラフィック，サウンドなどの機能が使えます．

1.3　動作モード

BASIC を起動すると（起動方法についてはユーザーズ・マニュアルを参照してください），PC-8801MKII は，画面に〝Ok〟という文字を出力し，BASIC はコマンドレベルになります．この状態のときは，あらゆる BASIC のコマンドをキーボードから入力できます．

1.3.1　ダイレクトモード

行番号をつけずに BASIC の文法に沿った文を入力した場合，その文は，キャリッジリターンを入力後，すぐに実行されます．これをダイレクトモードにおける実行といいます．

1.3.2　プログラムモード

行番号（1～65529：N - BASIC では 0 から）をつけて文を入力した場合，それはメモリの中にプログラムとして行番号とともに格納されます．そしていったん格納されたプログラムは，RUN コマンドおよび GOTO 文，GOSUB 文によって実行させることができます．これをプログラムモードによる実行と呼びます．

1.4　文

文とは，BASIC が行う手続きを記述している最小単位です．

文には，BASIC が実行する式，ステートメント，コマンド，関数などを書くことができます．また文は，コロン（：）を用いて他の文とつなぐことができます．これは複文（マルチステートメント）と呼ばれ，複文は，1行（行番号も含めて）

255文字以内の長さまで許されています.

1.5 行 番 号

BASICの各プログラム行は,行番号で始まらなければなりません.行番号には,1～65529まで(N - BASICは0から)の整数を用います.行番号はプログラム行をメモリに格納する順序を示し,実行も行番号の若い方から行われます.また分岐や編集の目印としても使用されます.

行番号の代りにピリオド(.)を使うことができる場合があります.ピリオドはLIST, AUTO, DELETEなどコマンド中で,エラー発生,編集などによってBASIC内のポインタが示している現在の行を表すものです.

```
LIST .
AUTO .
DELETE .-100
```

1.6 使用できる文字とコントロールキャラクタ

N_{88}(N)- BASICの使用できる文字は,英文字,カナ,数字,特殊記号,そしてグラフィックキャラクタより構成されており,これらはキャラクタセットと呼ばれます.英文字は大文字と小文字で,数字は0から9までです.これらキャラクタセットの詳細は,APPENDIXの"キャラクタ表"を参照してください.他にも特別の意味をもつコントロールキャラクタがあります.これについてもAPPENDIXを参照してください.

1.7 特殊記号の使い方

BASICでは,演算子(+,-,*,/)などの他にも特別な意味を持つ記号があります.ここでその意味をまとめて説明しておきます.

1. ピリオド(.)

   現在BASICが着目している行番号の値を持っておりポインタとして使用することができます.BASICが着目しているとは,新しい行を挿入した,エラーが発生したなどの行です.

   例) LIST .

2. ハイフン(-)

   LIST, DELETE命令など行の範囲を指定する時,何行から何行までという

場合に使います.

例)　DELETE　100 - 200

3. コロン（:）

マルチステートメントの区切りとして使います.

例)　A=B+C : PRINT　A

4. コンマ（,）

PRINT, INPUTなどパラメータが並ぶ場合その区切りとして多用されます.

例)　INPUT　A, B, C

　　　COLOR　7, , , 0

5. セミコロン（;）

PRINT 文などの区切りとして使います.

例)　PRINT　"A=" ; A

6. アポストロフィ（'）

REM 文（リマーク）の代用として使えます.

7. クエッションマーク（?）

PRINT 文の代用として使えます.

8. アスタリスク（*）

ラベル名の先頭に付けます.

9. スペース

BASIC は原則としてスペースを無視します. ただしコマンド, ステートメントの後には必ずスペースを入れなければなりません (N - BASIC では入れなくてもよい).

1.8　定　　数

1.8.1　定数の種類

N_{88} - BASIC の定数には以下のものがあります.

- 定数
  - 文字定数
  - 数値定数
    - 整数型
      - 10進形式
      - 8 進形式
      - 16進形式
    - 実数型
      - 単精度型
        - 固定小数点形式
        - 浮動小数点形式
      - 倍精度型
        - 固定小数点形式
        - 浮動小数点形式

1.8.2　文字定数

文字定数とは，255文字以下のダブルクォーテーション(″)で囲まれた英数字，カナ文字，記号などの列のことです．なお，(″)を文字列の中に記述する場合は，CHR$関数を用いなければなりません．

例)　　"Good Morning"
　　　"1 2 3 4 5 6 7 8 9"
　　　"パーナソル　コンピュータ"

1.8.3　数値定数

数値定数は，整数型，実数型に分けられ，それらは正あるいは負の数，または0です．

負の数の前には必ず符号をつけなければなりませんが，正の数の前の符号は省略することができます．

1.8.4　整　数　型

(1)　10進形式

－32768から＋32767までのすべての整数．または，数の後に％をつけたもの．小数点をつけることはできません．

例)　　32767
　　　－123
　　　32767％←─整数を表す．

(2)　8進形式

頭に＆Oまたは＆を伴った，0から7までの数字の並び．
＆0～＆177777の範囲

例)　　&12345(5349(10))
　　　&07777(4095(10))

(3)　16進形式

頭に＆Hを伴った，0からFまでの並び．

例)　　&H100(256(10))
　　　&HFFFF(65535(10))

注意： 8進形式または16進形式で入力された数値は，PRINT, LPRINTでは10進形式で出力されます．

例)　　10進数，８進数，16進数の関係

10 進数	8 進数	16 進数
0	&0	&H0
1	&1	&H1
2	&2	&H2
3	&3	&H3
4	&4	&H4
5	&5	&H5
6	&6	&H6
7	&7	&H7
8	&10	&H8
9	&11	&H9
10	&12	&HA
11	&13	&HB
12	&14	&HC
13	&15	&HD
14	&16	&HE
15	&17	&HF
16	&20	&H10
17	&21	&H11
18	&22	&H12
19	&23	&H13
⋮	⋮	⋮
60	&74	&H3C
61	&75	&H3D
62	&76	&H3E
63	&77	&H3F
64	&100	&H40
65	&101	&H41
66	&102	&H42
67	&103	&H43
68	&104	&H44
69	&105	&H45
⋮	⋮	⋮
250	&372	&HFA
251	&373	&HFB
252	&374	&HFC
253	&375	&HFD
254	&376	&HFE
255	&377	&HFF
256	&1000	&H100
257	&1001	&H101
258	&1002	&H102
259	&1003	&H103
⋮	⋮	⋮

1.8.5 実 数 型

実数型は，単精度型と倍精度型に分けられます．

1.8.6 単精度型

有効桁７桁の精度で格納されます．出力のときは７桁目が四捨五入され，６桁以下で表示されます．扱える数値は，－1.70141E＋38～1.70141E＋38です．

(1) ７桁以下の実数

(2) Ｅを使った指数形式

(3) 最後に！を伴った数

例）
```
1.23
-7.09E-06
3525.68
3.14!
```

1.8.7 倍精度型

有効桁16桁の精度で格納され，16桁以下で表示されます．

(1) ８桁以上の数

(2) Ｄを使った指数形式

(3) 最後に＃を伴った数

例）
```
1234567890
-1.09432D-06   +0.3141592653D+01
56789.0#
8657036.1543976
```

1.9　変　数

1.9.1　変数とは？

BASIC のプログラム中で使われる値を格納するためのエリアに，英数字から成る名前を対応させたものが変数です．

変数の値は，プログラマによって定義され，演算，参照などに使うことができます．これは演算，代入実行後，割り当てられます．数値変数は，値が割り当てられるまえに参照されたら 0 が代入され，文字変数はヌルストリング（空の文字列）が代入されます．

1.9.2　変数名と型宣言文字

N_{88}-BASIC において，変数名は最大40文字の英字で始まる英数字とピリオド（.）で表され，そのすべてを区別します．ただし，N-BASIC では最初の 2 文字までの判別しかしません．

例 1）

ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890123A

ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890123B

この 2 つは N_{88}-BASIC では区別される――

変数名は予約語（BASIC で使われるコマンド・ステートメントなどのキーワード）であってはなりませんが，予約語を含んだものは構いません．ただし，FN で始まる変数名は許されません．また英文字において大文字，小文字の区別はありません．

N-BASIC の場合は変数名中に予約語を含むことも許されません．

例 2）　RUN←変数とはならない

RUNI
XRUN　} ←―変数になれる

同じ変数名でも型が違えば区別されます．変数の型は型宣言文字によって決まり，型宣言文字は変数名の最後につけて，その変数の型を表します．型宣言文字を省略すると，"！" がついているとみなされます（単精度実数型変数となる）．

型宣言文字
- %　整数型
- !　単精度実数型
- #　倍精度実数型
- $　文字型

例3）　A
　　　A #
　　　A %
　　　A $

（A, A #, A %, A $）これらは区別されるが，AとA！は同じ．

変数の型を宣言するのにもう1つ便利な方法があります．それはDEFINT, DEFSNG, DEFDBL, DEFSTRの型宣言文をプログラム中で用いる方法です．これについては2章で詳しく説明します．

1.9.3　配列変数

配列とは，1つの変数名でいくつかの要素を参照することのできる変数です．配列変数のそれぞれの要素は，整数または整数表記による添字によって参照されます．

配列変数の次元は255次元まで，添字はメモリの範囲内で許されており，これらの大きさはDIM文（2章参照）で宣言します．ただし，各添字は原則として0から始まりますから，実際の要素数は添字の数＋1となります．

例）	DIM A(10)	1次元配列，要素の数＝11
	DIM TA(10, 50)	2次元配列，要素の数11×51＝561
	DIM NAME$(2, 5, 3)	3次元配列，要素の数3×6×4＝72

注：各添字の数が10以下の時はDIM文による宣言を省略することができます．

1.9.4　予約変数

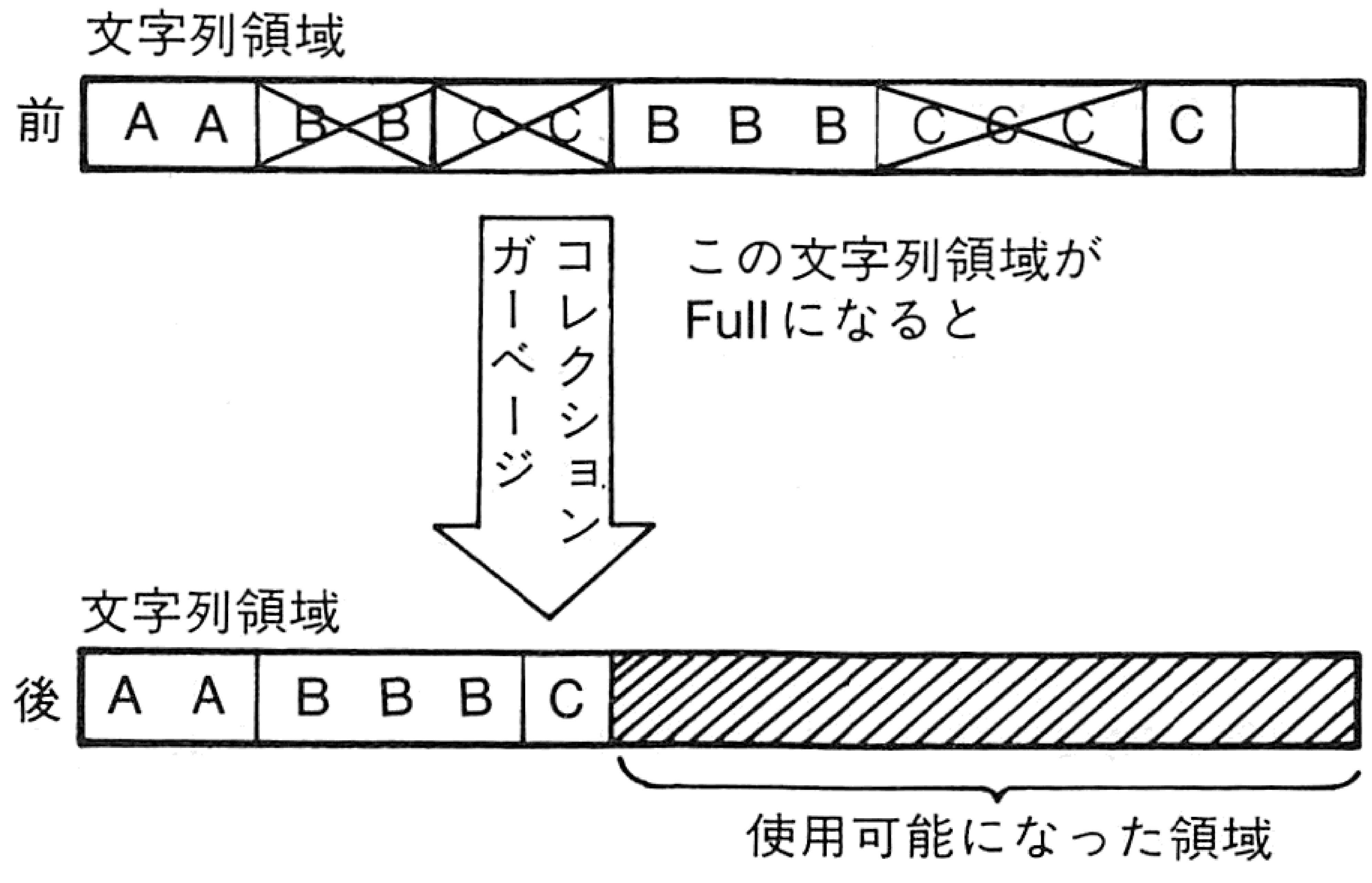
N_{88}(N)-BASICには，BASIC自身が専用に使う予約変数があります．これらの変数は，ユーザーによって一般の変数として使うことはできません．

TIME$	現在の時分秒をHH：MM：SSの形でもっています．同じ形で代入することもできます．
DATE$	現在の年月日をYY／MM／DDの形でもっています．同じ形で代入することもできます．
ERL	エラーの生じた時，エラーが発生した行番号をもっています．代入することはできません．
ERR	エラーが発生した時，生じたエラーのエラーコードをもっています．代入することはできません．

1.9.5　ガーベージコレクション

例えば，文字変数A$，B$，C$にそれぞれ文字例AA，BB，CCを代入すると，これらは，順番に文字列領域内に書き込まれます．次にまた，B$，C$にBBB，CCCを代入すると，初めにB$，C$が取った領域に書き込むのではなく，新たな領域にB$，C$の内容を書き込みます． 更に，C$にCを代入するとまた新たな領域に書き込む事になります． ここで，現在，A$，B$，C$の持っている文字列は，A$はAA，B$はBBB，C$はCを持っています．しかし文字列領域内には，これらが順番に存在するものではなく，あちこちに存在していて間には不要になった文字列があります．そして，このように次々に文字変数に文字列が与えられ，文字列領域が**Full**の状態になると，現在使われている文字列の領域と不要になった文字列がある領域と2つに分けて，使用可能な領域を確保する事をガーベージコレクションと言います．

概念図



1.10　型変換

N_{88}(N)-BASICの数値データは，必要に応じてその型から他の型に変換することができます．ただし，文字型と数値型の間でこの変換を行うことはできません．

(1)　ある型の数値データが，違った型の数値変数に代入された場合，数値は，その変数名によって宣言された型に変換されます（N と N_{88} では実数から整数への変換の仕方が異なりますから注意してください）．

例)
```
10  ABC%=1.234
20  PRINT  ABC%
RUN
 1
```

(2) 精度の違う数値間の演算の場合，精度の高い方に変換されて，演算が行われます．たとえば，10#／3の場合は，10#／3#として演算が行われます．

例)
```
10  A#=10#/3
20  B#=10#/3#
30  PRINT  A#, B#
RUN
3.333333333333333     3.333333333333333
```

(3) 論理演算の場合，扱われる数値はすべて整数に変換され，結果は整数で与えられます（N と N_{88} では実数から整数への変換の仕方が異なりますから注意してください）．

例)
```
10  A#=12.34
20  B=NOT  A#
30  PRINT  B, A#
RUN
-13              12,34000015258789
```

(4) 実数が整数に変換される場合は，小数点以下は四捨五入されます（N-BASIC では切り捨てます）．この時，整数型で扱える範囲を超えた場合はエラーが起こります．

例)
```
10  A%=34.4
20  B%=34.5
30  PRINT  A%, B%
RUN
34              35
```

```
10  A#=1.234E+07
20  B%=A#
30  PRINT  B%, A#
RUN
Overflow
```

(5) 倍精度変数が単精度変数に代入された時は，変数の値は有効数字 7 桁に丸めたものとなります．単精度変数の精度は 7 桁であり，もとの倍精度の数値との誤差の絶対値は，5.96E−8 以下となります．

例)
```
10  A#=1.23456789#
20  B!=A#
30  PRINT  A#, B
RUN
1.23456789          1.23457
```

1.11　式と演算

式とは定数や変数を演算子で結合した一般的な数式や，単に文字や数値，あるいは変数だけのものをいいます．

例）　"BASIC"

　　　3.14

　　　10+3／5

　　　A+B／C−D

　　　TAN (DO)

BASICの演算は次の5つに分類されます．

1. 算術演算
2. 関係演算
3. 論理演算
4. 関数
5. 文字列演算

1.11.1　算術演算

算術演算子には次のようなものがあります．

実行順序	演算子	演算	例
↓	∧	指数演算	X∧Y
	−	負号	−X
	＊，／	乗算，実数の除算	X＊Y，X／Y
	＋，−	加算，減算	X＋Y，X−Y

演算の実行順序を変更する場合カッコを使用します．カッコの中の演算子は他の演算子より先に実行されます．カッコ内においては通常の実行順序に従います．

次に実行例を示します．

	代数表記	BASICの表記
1）	$2X+Y$	2＊X＋Y
2）	$\frac{X}{Y}+2$	X／Y＋2
3）	$\frac{X+Y}{2}$	(X＋Y)／2
4）	X^2+2X+1	X∧2＋2＊X＋1
5）	X^{Y2}	X∧(Y∧2)
6）	$(X^Y)^2$	X∧Y∧2

7)　Y(−X)　　　　　Y＊(−X)

(1) 整数の除算と剰余の計算

整数の除算は¥によって行われます．扱われる数値が実数の場合は，演算が実行される前に小数点以下が四捨五入されます．商は小数点以下が切り捨てられた整数となります．

例)　10¥3＝3　　　　(10／3＝3... 1)

　　　23.75¥5＝4　　(24／5＝4... 4)

剰余の計算はMODによって行われます．扱われる数値が実数の場合は，演算が実行される前に小数点以下が四捨五入されます．結果は整数の割り算の余りです．

例)　13.3 MOD 4＝1　　　　(13／4＝3... 1)

　　　25.68 MOD 6.99＝5　　(26／7＝3... 5)

(2) 0での除算

式の実行時に0での除算が行われた場合は，エラーメッセージを出力しますが，結果は計算機が扱うことのできる最大の数とみなし，それを除算の結果として処理を続行します．

また，べき乗の実行時に，0に対して負のべき乗を行った場合も同様になります．

例)
```
PRINT 2/0
/0
 1.70141E+38

PRINT 0^-1
/0
 1.70141E+38
```

(3) 桁あふれ（オーバーフロー）

代入や演算の結果がその変数の型内で表現することのできる範囲をこえた場合，桁あふれが発生します．

桁あふれが起こった場合，BASICは〝Overflow〞(OV Error)エラーを出力し，BASICが扱うことのできる最大の数を結果として与え，処理を続行します．

例)
```
PRINT 3 ^ 300
OV
 1.70141E+38
```

1.11.2 関係演算

関係演算子は２つの数値を比較するときに用います．結果は，真(－１)，偽(０)で得られ，条件判定文などプログラムの流れを変えるのに用いられます (IF 文参照)．

関係演算子	内容	例
＝	等しい	X＝Y
<>，><	等しくない	X<>Y，X><Y
<	小さい	X<Y
>	大きい	X>Y
<＝，＝<	小さいか等しい	X<＝Y，X＝<Y
>＝，＝>	大きいか等しい	X>＝Y，X＝>Y

注意：＝は代入文にも使うので注意すること．

IF 文の中での使い方の例を次に示します．

```
IF  X=0  THEN 1000
IF A+B < > 0 THEN X=X+1 : Y=Y+1
```

1.11.3 論理演算

論理演算子は複数の条件を調べたり，ビット操作やブール演算を行ったりするのに用います．論理演算は，ビットごとに０または１を結果として与えます．

各論理演算の内容を次に示します．

NOT ＝ not（否定）

X	NOT X
1	0
0	1

AND ＝ and（論理積）

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR ＝ inclusive or（論理和）

X	Y	X OR Y
1	1	1
1	0	1

0	1	1
0	0	0

XOR = exclusive or（排他的論理和）

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

IMP = implication（包含）

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

EQV = equivalence（同値）

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

論理演算子も関係演算子のように，プログラムの流れを変えるのに用いられます．この場合，論理演算子は２つ以上の関係演算子と結ぶことができます．

例）　(1) IF X<0 OR 99<X THEN 1000
　　　(2) IF 0<X AND X<100 THEN X=0
　　　(3) IF NOT (A=0) THEN 20

(1) Xが負または，99より大きければ，行番号1000へ飛ぶ．
(2) Xが正でかつ，100より小さければ，Xに０を代入する．
(3) Aが０でなければ，行番号20へ飛ぶ．

注意：論理演算子は演算の前に扱う数値を－32768から＋32767までの２つの補数表示の整数に変換します．もし，この範囲外となれば〝Overflow〟(OV Error)エラーとなります．もし，０（偽）と－１（真）しか与えられなかったなら，論理演算子は０と－１しか結果として与えません．

指定された論理演算では，この整数に対しビットごとに演算を行います．したがって，論理演算子はバイトデータをあるビットパターンに照らし合わせて調べ

ることができます．

たとえば，AND 演算子は機器の I／O ポートのステータスバイトの必要なビット以外のすべてのビットをマスクすることに使えます．また OR 演算子はある 2 進数を作るために 2 つのビットパターンを混在させることができます．

以下の例は論理演算子がどのように働くかの例です．

63　AND　8＝8　　$63=(111111)_2$，$8=(001000)_2$
　したがって，63　AND　$8=(001000)_2=8$

−1　AND 8＝ 8　　$-1=(1111111111111111)_2$，$8=(001000)_2$
　したがって，−1　AND　8＝8

12　OR　11＝15　　$12=(1100)_2$，$11=(1011)_2$
　したがって，12　OR　$11=(1111)_2=15$

32767　OR −32768＝−1
　$32767=(0111111111111111)_2$
　$-32768=(1000000000000000)_2$
　したがって，32767　OR　−32768＝
　$(1111111111111111)_2=-1$

12　XOR　11＝7　　$12=(1100)_2$，$11=(1011)_2$
　したがって，12　XOR　$11=(0111)_2=7$

10　XOR　10＝0　　$10=(1010)_2$
　したがって，10　XOR　$10=(0000)_2=0$

NOT　X＝−(X＋1)　　任意の数の補数表示は 1 の補数に 1 を加えたものである．

1.11.4 関　　数

関数とは与えられたある引数に対して，ある決った演算を行うもので，値としては，この演算の結果を返します．

BASIC は〝組み込み関数〟として SIN（正弦），SQR（平方根）などの数値関数や CHR$, LEFT$などの文字列関数を本体内にもっています．これらの関数については 3 章で詳しく説明します．

また，BASIC は〝ユーザー定義関数〟としてユーザーが自由に定義できる関数機能ももっています．これは 2 章の〝DEF FN〟の項で説明します．

また，これらの初等関数(SIN 関数など)は，引数が倍精度のときは倍精度* となり，引数が整数や単精度のときは単精度となります．

＊倍精度関数機能はN_{88}DISK-BASICのみ使用可能。

一般に引数に整数しかとらないものは，小数部分を四捨五入して整数に丸めてから演算を行います（N - BASIC では切り捨てます）.

次にその使い方の例を示します.

```
A=SIN(3.14)+COS(3.14)
PRINT 2, 2 * 2, SQR(2)
```

1.12 文字列の演算

1.12.1 文字列の連結

文字列は演算子 + によって連結することができます.

例）

```
10  A$= "NEC" : B$= "PC -" : C$= "8801MK2"
20  D$=A$+ "  "+B$+C$
30  PRINT D$
RUN
NEC PC-8801MK2
```

1.12.2 文字列の比較

文字も数値の比較に用いられるものと全く同じ関係演算子を用いて比較することができます.

=, <, >, <>, ><, <=, =<, >=, =>

文字列の場合それぞれの文字列の最初から１文字ずつ,文字の比較を行います.もし相互に全く同じ文字列の場合は，その２つの文字列は等しくなりますが，１箇所でも違った場合は,その文字のキャラクタコード(APPENDIX のキャラクタ参照）の大きい方の文字列が大きくなります．文字列の片方が短かくて比較が途中で終った場合は，短かい文字列の方が小さくなります.

文字列の比較においては空白なども意味をもちますから注意してください.

例）

```
"AA" < "AB"
"BASIC" = "BASIC"
"X&" > "X #"
"PEN   " > "PEN"
"cm" > "CM"
"DESK" < "DESKS"
```

このように，文字列の比較は文字列の内容を調べたり，文字をアルファベット順に並べたり（ソート）することに使うことができます.

1.13 演算の優先順位

演算は次の順位によって行われます.

1. カッコで囲まれたもの
2. 関数
3. 指数（べき乗）∧
4. 負号（－）
5. ＊，／
6. ￥
7. MOD
8. ＋，－
9. 関係演算子（<，>，＝など）
10. NOT
11. AND
12. OR
13. XOR
14. IMP
15. EQV

1.14 数値演算における誤差

1.14.1 誤　差

コンピュータは絶対に正確だと思っている方が大多数だと思われますが，次のように必ずしも正確であるとはいえないことがあります.

例）　1. 実数型変数のプログラム

```
100 C＝0：A＝0
110 PRINT "C＝";C, "A＝";A
120 IF A＝1 THEN 150
130 C＝C＋1：A＝A＋0.1
140 GOTO 110
150 END
```

このプログラムはカウントCと変数Aを0にしてから，変数Aが1となったところで終わるものです．カウントCは1ずつ，変数Aは0.1ずつ増えていきます．このプログラムを実行してみましょう.

```
RUN
C=0          A=0
C=1          A=0.1
C=2          A=0.2
C=3          A=0.3
C=4          A=0.4
C=5          A=0.5
C=6          A=0.6
C=7          A=0.7
C=8          A=0.8
C=9          A=0.9
C=10         A=1
C=11         A=1.1
C=12         A=1.2
C=13         A=1.3
```

不思議なことに，A=1.0となってもプログラムは止まりません．

例） 2．関数のプログラム

```
100 S=15
110 FOR Y=0 TO S
120 A=2^Y
130 PRINT "2^";Y;"=";A
140 NEXT Y
150 END
```

このプログラムは 2^0 から 2^{15} までを計算してプリントするものです．実行させてみましょう．

```
RUN
2^0 =1
2^1 =2
2^2 =4
2^3 =8
2^4 =16
2^5 =32
2^6 =64
2^7 =128
```

2＾8　＝256

2＾9　＝512

2＾10＝1024

2＾11＝2048

2＾12＝4096

2＾13＝8192.01

2＾14＝16384

2＾15＝32768

ここで，2＾13に注目してください．8192.01というように，小数点以下2桁まで表示されていますね．

例）　3．入力のプログラム

```
100 INPUT  A#（注）
110 PRINT  A#
120 END
```

注）　#は変数Aが倍精度変化として扱われることを表わしています．

このプログラムを実行してみましょう．“?”マークが表示されたら，0.0075とリターンキーを入力します．

```
RUN
? 0.0075
7.4999999999999999D-03
```

例）1のプログラムでは，A＝1.0となっているのにプログラムが止まらず，(例2）と（例3)のプログラムでは，予想に反した結果が表示されます．その理由を簡単に説明すると次のようになります．

例）1のプログラムでは，A＝1.0と表示された時に，実は，ちょうど1になっていないからです．この現象は，コンピュータにおける数値の内部表現形式が原因となって起こります．後でもう少し詳しく説明します．

例）2のプログラムで小数点が現われたのは，関数の値を近似計算式で求めているためです．そして，他の関数（SIN, COS, TAN, etc）についても同じように近似計算を行っているための誤差が含まれて出力されます．

例）3の場合は，入力されたデータを内部表現形式に変換する場合に生まれる誤差の影響です．

1.14.2　対　策

それでは，どうすれば誤差の影響の少ないプログラムになるでしょうか．主な

対策をあげてみましょう.

a. 定数型変数を比較する場合は等号(＝)ではなく，不等号(＞，＜)を使用します.

b. FOR～NEXT ループの変数には整数型変数を使用します.

c. 結果を整数値として表示する場合は，INT 関数や FIX 関数を使用します.
このとき注意しなければならないのは，理論上，結果が

```
A=1
```

となるものが

```
A=0.9999………
```

となっている場合です．このとき，このままでINT 関数を使用すると，

```
PRINT INT (A)
0
```

となり，結果が 0 になってしまう時があります．これは，IN T関数が与えられた引き数の整数部分だけを取り出すからです．このような場合は，INT 関数に代入する前に，0.1を加えておくという手段が有効です.

d. 精度の高い関数計算を必要とする場合は，N_{88}DISK-BASIC で倍精度関数を利用します.

e. 結果を画面やプリンタに出力する場合は，PRINT USING 文を利用します.

例 1 の場合には，aに従い，行番120の文を次のように変更します．これで小数点以下 4 桁の精度を必要とする場合の比較を行なうことができます.

単精度の数値は，有効桁 6 桁，倍精度は有効桁16桁ですので，このことを考慮して精度を決定して下さい.

```
120 IF 0.9999<A AND A<1.0001 THEN 150
```

1.14.3 数値の内部表現形式

一般にコンピュータ内部での数値の演算方式には次の 2 通りの方式があります.

2 進演算方式……入力された数値を計算機内部で 2 進数に変換した後，いろいろな演算を行う.

例) 入力値 1 2 3

内部値 1 1 1 1 0 1 1 (2)

10進演算方式……入力された数値を10進数のまま計算機内部に記憶し，演算を行う.

例) 入力値 1 2 3

内部値 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 (2)

→ 1 2 3 (10)

注） (2)は2進数を表わし，(10)は10進数であることを表しています．

これら2つの演算方式の大きな違いは，数値を2進数に変換するところです．つまり，10進数の形で残すか，2進数の形に変換してしまうかによって，再び10進数として表示する場合に誤差が生まれるかどうかが決ってきます．

例えば，0.83を2進演算用に変換すると次のようになります．

$$0.83=1/2^1+1/2^2+1/2^4+1/2^6+1/2^{10}+1/2^{11}+1/2^{12}+\cdots\cdots$$

従って，計算機内部では，

1101010001111………

という形になります．しかし，計算機のメモリには限度があり，通常は32ビットとか64ビットというように制限されています．従って再び10進数に変換すると，前とは異った値になります．

ですから，2進演算方式を採用しているPC-8801MKⅡでは，多少の誤差が現われますが，これは特に異常ということではありません．

ではなぜ多くのパーソナルコンピュータが2進演算を採用しているのでしょうか．主な理由は次の2つです．

1. 演算を早く行う．
2. 大きな数値でも少ない桁数で表現できる．

例） 255

10進演算

00000010 00000101 00000101

（3バイト）

2進演算

11111111

（1バイト）

1.15 画面モード

ここでいう画面モードとは，文字を表示するテキスト画面の表示桁数，ドットグラフィックスの分解能，白黒モードかカラーモードかなど画面に対する操作のすべての意味を含むものです．N_{88}(N)-BASICは，この画面モードを必要に応じて選ぶことができます．

ただし，N_{88}-BASICとN-BASICの画面モードは次の様に異っています．

1.15.1　N-BASIC の画面モード

（1）　文字表示桁数とドットグラフィックスの分解能

N-BASIC で許される画面の表示行数は，20行と25行です．また1行に表示できる桁数は，80，72，40，36桁が許されています．

N-BASIC のグラフィックスは，キャラクタを横2ドット，縦4ドットと考えたセミグラフィックで，根本的に文字を表示するテキスト画面とドットグラフィックスの画面を区別していません．従ってドットグラフィックスの分解能は，必然的にキャラクタの表示桁数（表示行数）に依存します．例えば80×25の画面モードの場合，(80×2)×(25×4)⇨160×100の分解能となります．その他の表示桁数の場合も同様に計算できます．次の図は表示文字数とドットグラフィックスの分解能の関係です．これらの設定の仕方は後述のWIDTHを参照してください．

表示文字数(横×縦)	分　解　能(横×縦)
36×20	72×80
40×20	80×80
72×20	144×80
80×20	160×80
36×25	72×100
40×25	80×100
72×25	144×100
80×25	160×100

テキスト画面とグラフィック画面

（2）　白黒モードとカラーモード

N-BASIC はどの文字表示桁数の時でも，白黒モードとカラーモードを選ぶことができます．

カラーモードの場合，8色のカラーを文字，ドット共に使うことが可能です．ただし，前記したようにN-BASIC のドットグラフィックスは，キャラクタに依存するセミグラフィックス方式ですから，1キャラクタ内の複数のドットを違う色で指定することはできません．また1行中で，20回以上違う色を指定しても20回目からは，20回目に指定した色になります．

白黒モードの場合は，ブリンクやリバースなど8つのファンクション(機能)をキャラクタ単位に設定することができます．

これらの実際の設定の仕方は後述のCOLOR，CONSOLE を参照してください．

1.15.2　N88-BASIC の画面モード

（1）　テキスト画面とグラフィック画面

N88-BASIC では，リストなどキャラクタ文字を表示するテキスト画面とドットグラフィックスを表示するグラフィックス画面を完全に区別しています．従って個別にそれらのモードを設定する命令があり，片方のモードを変化させても，もう片方が影響を受けることはありません．

（2）　テキスト画面

テキスト画面の表示文字数は，横80桁か40桁で，縦20行か25行を選択できます．また画面が前記のどのモードであっても，カラーと白黒を切り替えて使うことが可能です．このテキスト画面の機能及びその設定はN-BASIC のテキスト画面の場合と同様ですから，1.15.1の〝N-BASIC の画面モード〃を参照してください．ただし，N-BASIC が持っているセミグラフィック機能はありません．

表示文字数(横×縦)
40×20
80×20
40×25
80×25

テキスト画面

（3）　グラフィック画面

N88-BASIC のグラフィック画面は大量のメモリ（48K）によって構成されており種々のモードを選ぶことができます．ここで，それぞれのモードについて説明する前にこのグラフィック画面（メモリ）がどのような構成になっているのか説明しておきましょう．

分解能(横×縦)	備　　考
640×200	カラーまたは白黒 3 ページ
640×400	白黒 1 ページ

グラフィック画面

次の図はそのグラフィック画面の基本構成で，このように640×200の分解能を持つ画面が 3 枚用意されています．このそれぞれの画面はプレーンまたはページと呼ばれます．N88-BASIC はこれら 3 つの画面をアレンジすることにより，色々な画面モードを作り出しているのです．

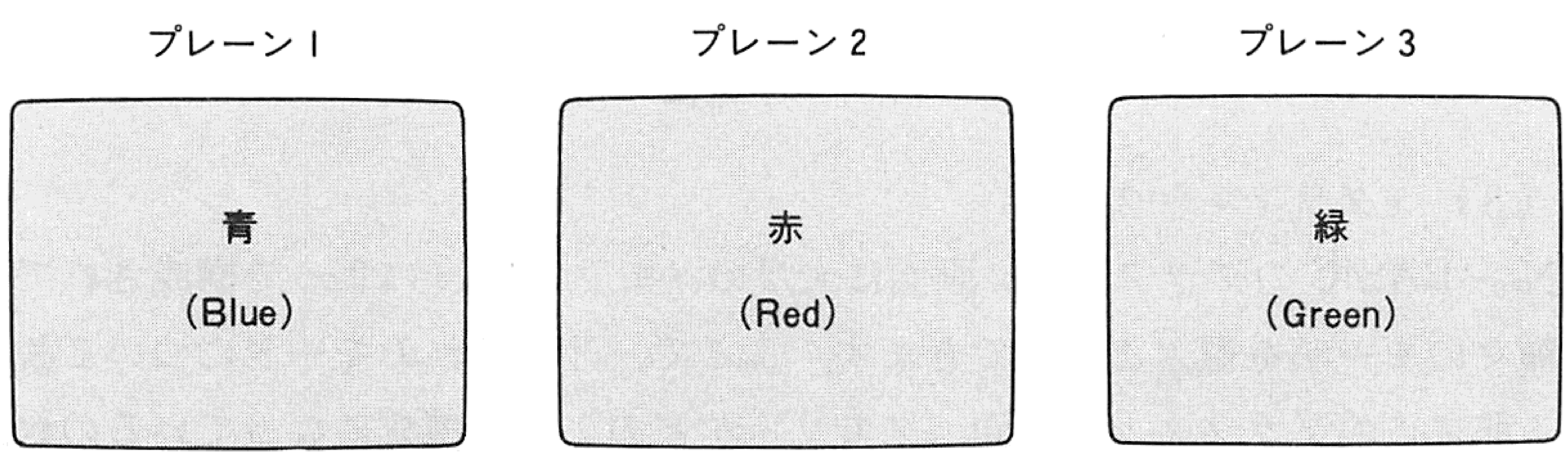


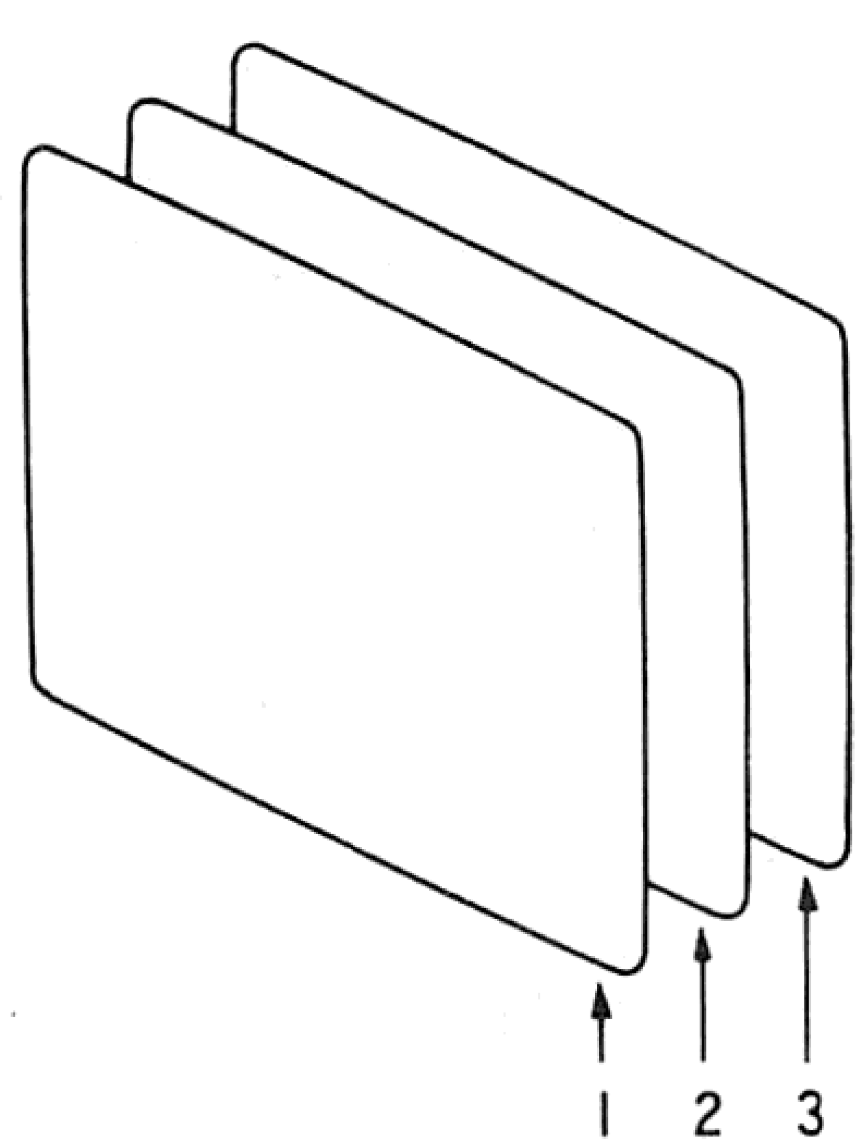
ではこのプレーンの説明を交えながら，それぞれのモードについて解説します．

カラーモード

640×200の分解能で各ドットごとに8色のうちから任意の一色を指定することができます．

このモードの時，各プレーンは次の図のように1のプレーンが青，2のプレーンが赤，3のプレーンが緑というように決められています．そしてこの3枚のプレーンはRGB（Red, Green, Blue）合成されて画面上に表示されます．例えば青のプレーンと赤のプレーンにドットがあり，緑のプレーンには何もない場合は，青と赤の合成によりそのドットは紫で表示されます．またすべてのプレーンにドットがあった場合は白で表示されることになります．





カラーモード

次の図はこれらのプレーンと各カラーコードの関係です．

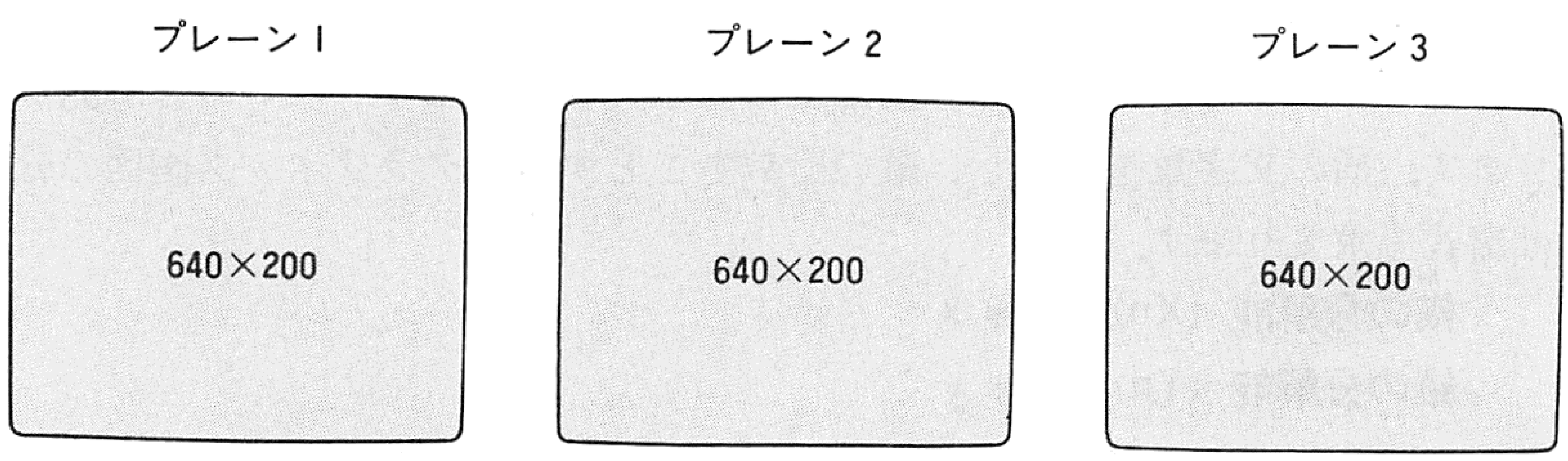
〈色〉	緑のプレーン（ビット2）	赤のプレーン（ビット1）	青のプレーン（ビット0）	〈カラーコード〉
黒	0	0	0	0
青	0	0	1	1
赤	0	1	0	2
紫	0	1	1	3
緑	1	0	0	4
水色	1	0	1	5
黄色	1	1	0	6
白	1	1	1	7

このように青のプレーンがビット 0，赤のプレーンがビット 1，緑のプレーンがビット 2 というように 3 ビットに対応しており，この 3 ビットによって表現される 2 進数の10進表記がカラーコードなのです．

以上の事を理解しておけば，後述するパレット，PAINT，PUT など N_{88}-BASIC の強力なカラーグラフィック命令を十分に使うことができます．

白黒モード

640×200の分解能の白黒画面が 3 ページあり，それぞれを単独で表示したり合成したりすることができます．プレーンの構成は次の図のようにそれぞれが白黒画面です（白黒モードの場合プレーンのことを特にページと呼びます）．

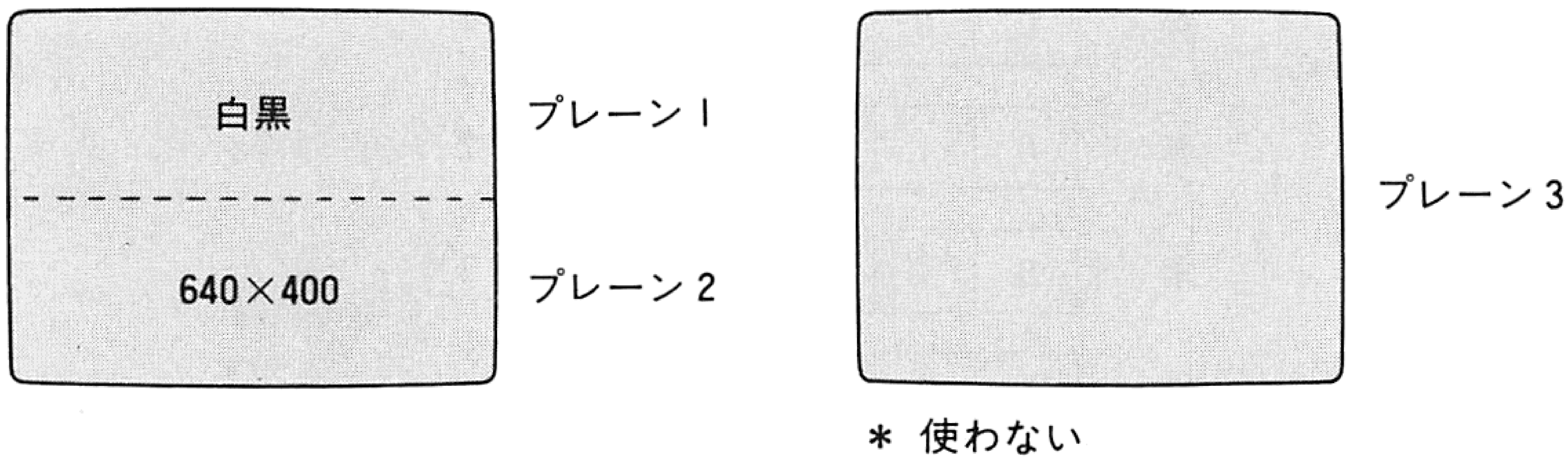


白黒モード（3ページ）

これらのページは，後述するSCREEN命令でアクティブページ(書き込むページ）とディスプレイページ（表示するページ）を適当に選ぶことによって複数のページを合成したり，表画面を表示している間に裏画面にだけ書き込むなどの特殊な使い方が可能です．

高分解能白黒モード

640×400の分解能で白黒画面が1ページのモードです．



高分解能白黒モード

プレーンの構成は図のように，プレーン1とプレーン2が2つ上下につながった形となっています．このモードではプレーン3は使いません．

但しこのモードは専用の高分解能モニタを使用した時のみ設定可能となります．詳しくはユーザーズ・マニュアルを参照ください．

1.16　ディスプレイ画面上の座標系

1.15の画面モードの所で説明したように，N－BASICとN_{88}－BASICでは画面上の分解能が異なっています．従っておのずとそれぞれの座標系も異なってきます．次に説明するのは各BASICでの画面上の座標系です．

1.16.1　N－BASICの座標系

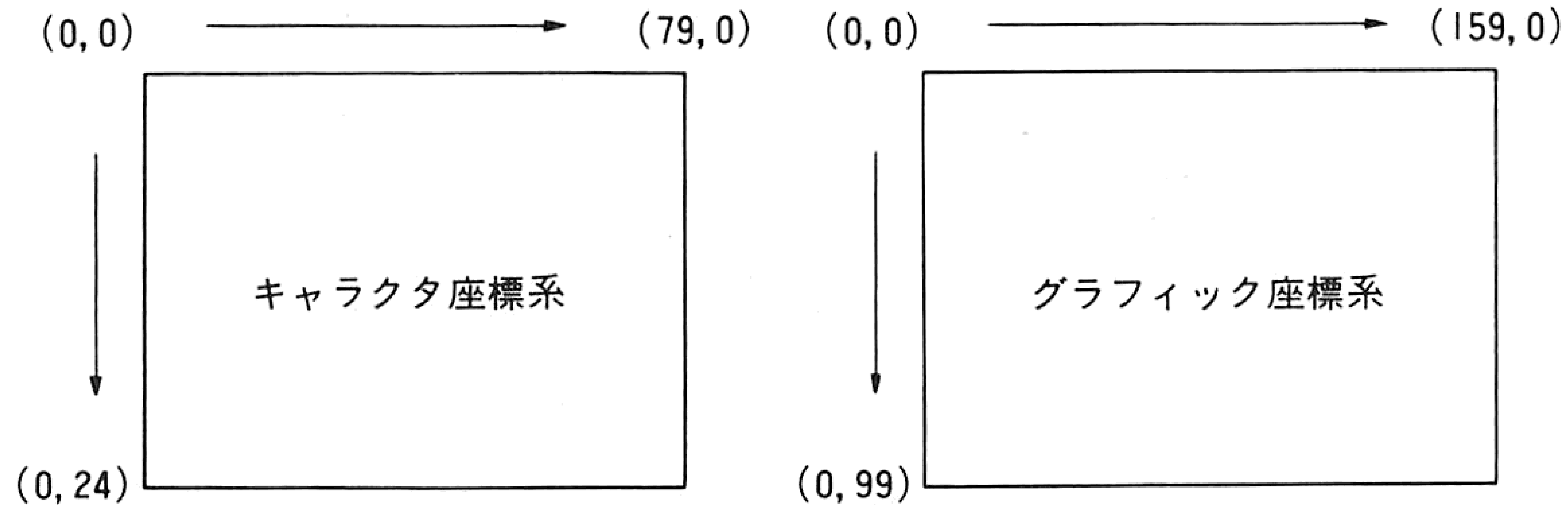
N－BASICのドットグラフィックスは，前記したようにキャラクタに依存するセミグラフィック方式ですから，画面上の表示文字数によりドットの分解能が決まります．横の文字数をxとし，縦の行数をyとするとグラフィック画面の分解能は次式で求まります．

横の分解能（Xn）＝2 ＊ x

縦の分解能（Yn）＝4 ＊ y

次に各座標の画面に対する位置ですが，キャラクタ座標，グラフィック座標とも画面の左上が原点で（0，0）です．80×25の文字数を設定したとすると各座標

の位置は次の図のようになります．座標の最大値は，原点が（0，0）から始まることにより，必ず各分解能－1という値になります．



80×25設定時の各座標系

1.16.2 N_{88}-BASICの座標系

N_{88}-BASICの座標系も，キャラクタ画面，グラフィック画面とも画面の左上が原点(0，0)であり，基本的な考え方はN-BASICと同様です．ただし，テキスト画面とグラフィック画面が分離されていて，相互干渉することはありません．

（1） キャラクタ座標系

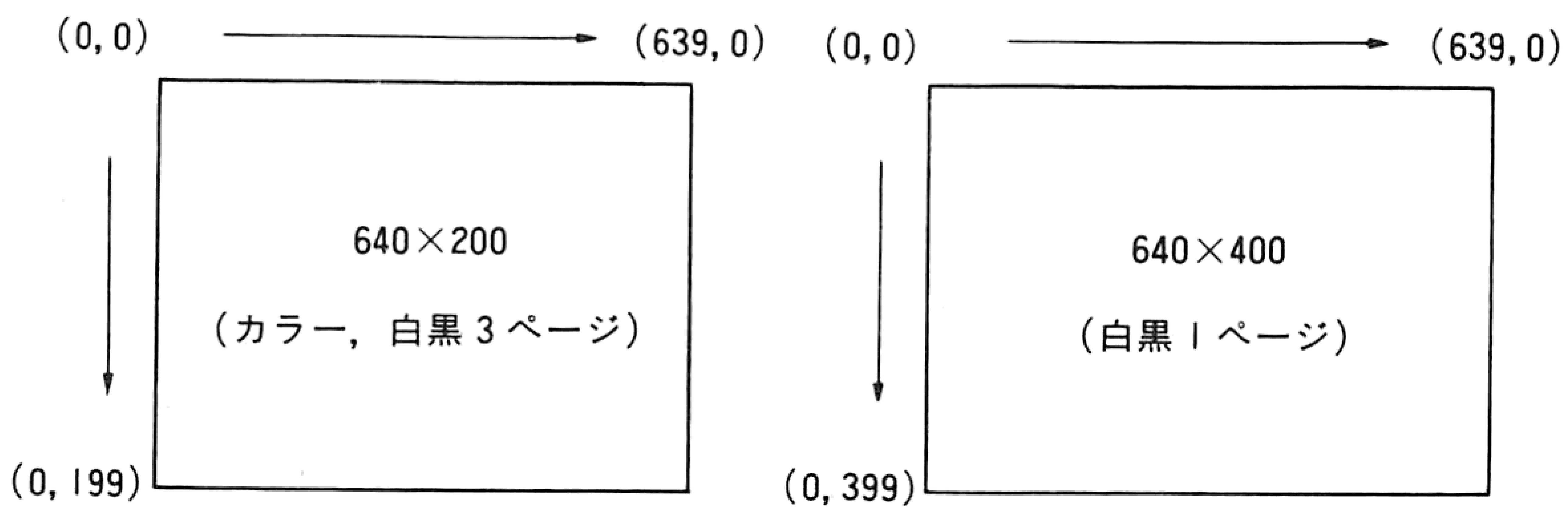
キャラクタ座標系はテキスト画面に展開される座標系です．

キャラクタ座標系は，横の文字数として36文字と72文字が指定できないことを除けば，N-BASICと全く同じです．

（2） グラフィック座標系

グラフィック座標系はグラフィック画面に展開される座標です．

N_{88}-BASICのグラフィックスの分解能は，640×200と640×400の2つですから各モードの座標系は次の図のようになります．



N_{88}-BASICのグラフィック座標系

1.17 ウィンドウとビューポート（N88-BASIC）

1.17.1 ワールド座標系とスクリーン座標系

1.16の〝ディスプレイ画面上の座標系〟の所で，N₈₈- BASICはキャラクタ座標系とグラフィック座標系があることを説明しましたが，グラフィック座標系はさらにスクリーン座標系とワールド座標系という2つの座標系を持っています．

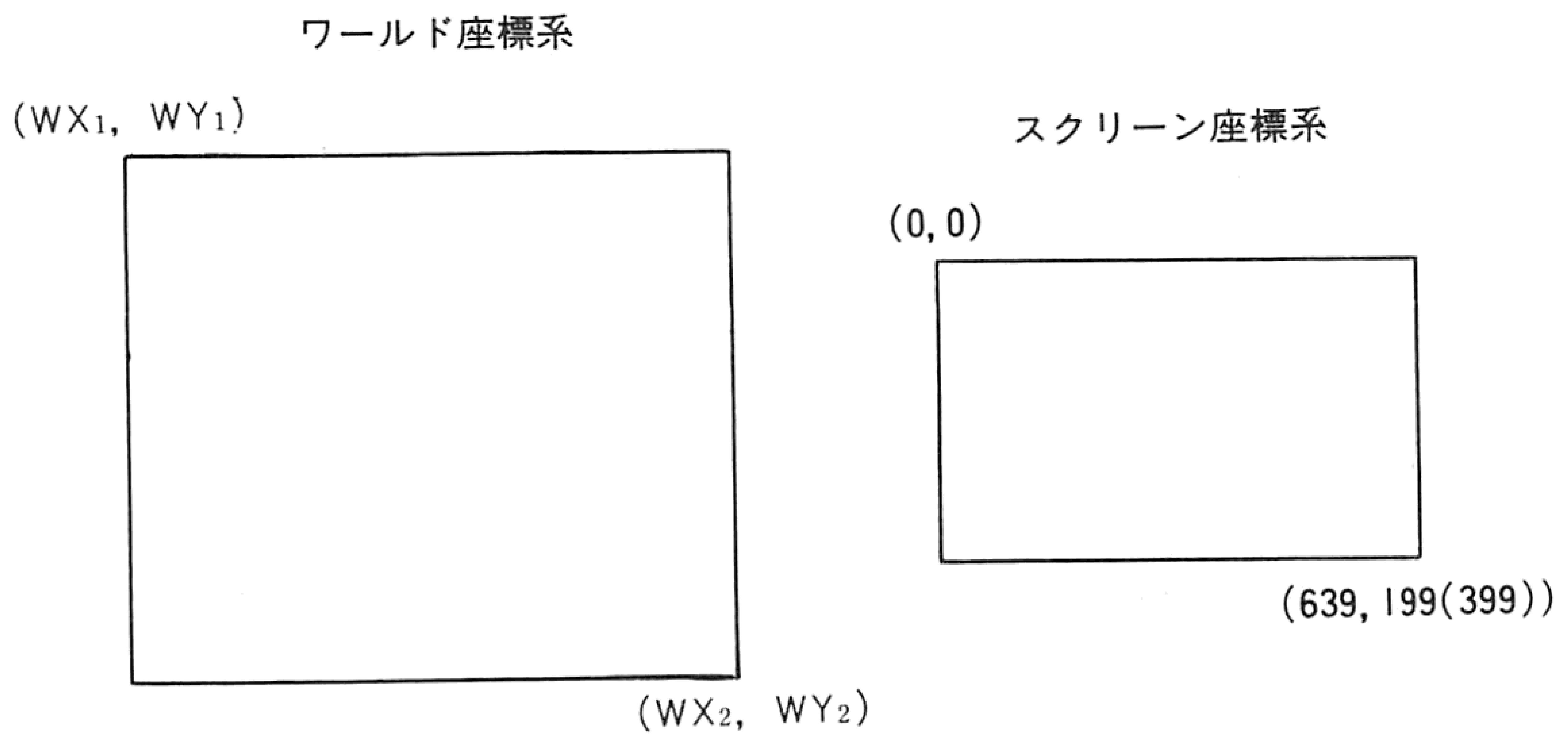
この2つの座標系は，グラフィック画面に対してのみ意味を持つもので，N-BASICなどが持っていた固定された画面上の座標系（物理的座標）だけでなく，仮空の大きな座標系（論理的座標）を扱うために必要なものです．これは本格的なグラフィックス処理の概念で，N₈₈- BASICで初めて採用されました．

まずスクリーン座標系とは先程説明したグラフィック座標上に存在するもので座標上の各点は画面の1ドットに対応します．スクリーン座標については，1.17.3ビューポートで説明します．現時点ではグラフィック座標と同等なものと考えておいてください．

次にワールド座標系とは，ユーザーが使うことのできる最大の座標系であり，BASICが設定した仮空の座標です．その大きさは縦・横とも−1.70141 E+38〜1.70141 E+38まで許されていて，負の領域座標も表現することができます．

但し，水平方向の幅(WX_2-WX_1)及び垂直方向の幅(WY_2-WY_1)も1.70141 E+38を越えてはなりません．後に解説する，点を打つ(PSET)，線を引く(LINE)などのグラフィック命令のほとんどはこのワールド座標系に対して働きます．

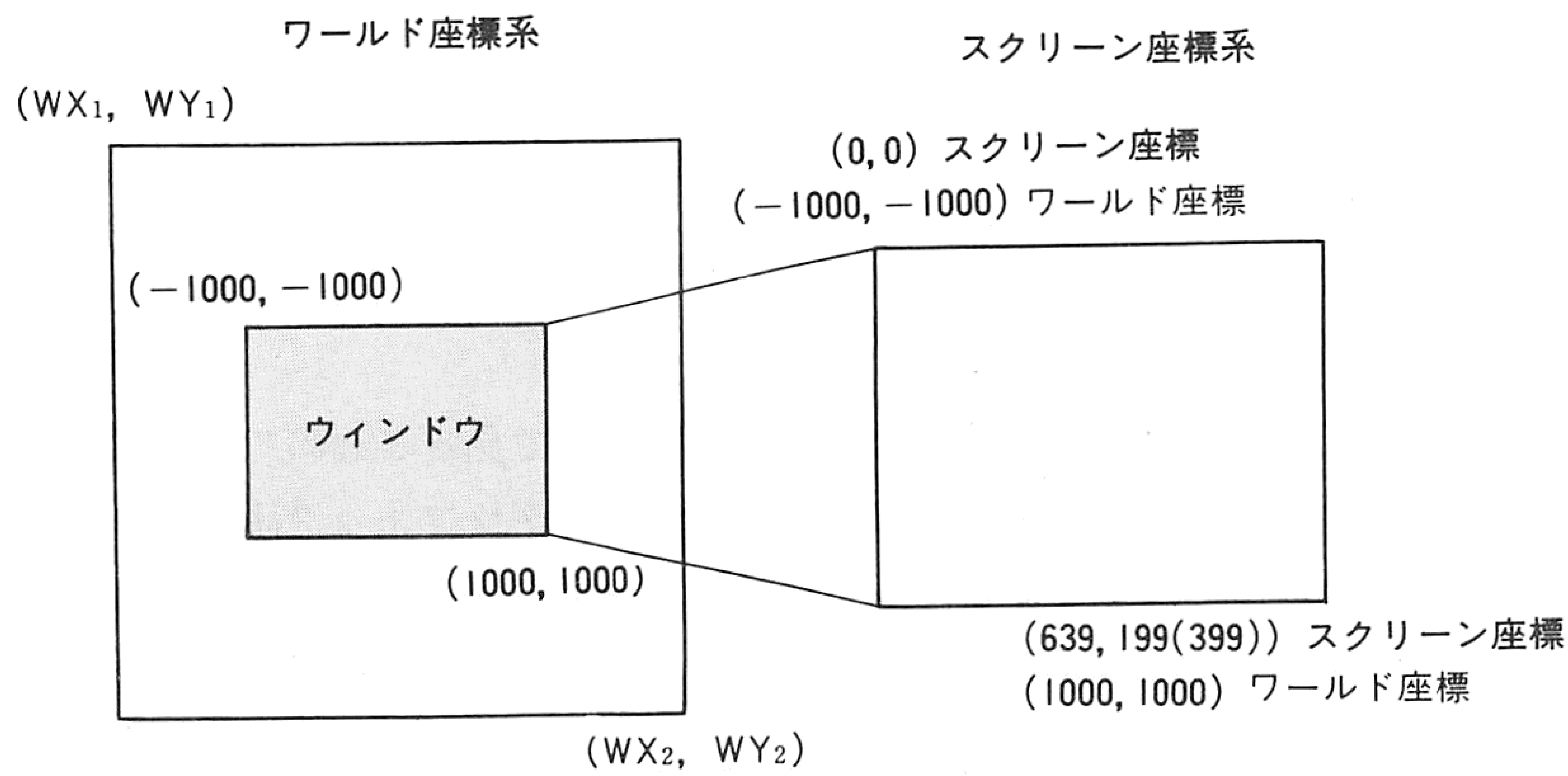
次の図はこの2つの座標系です．



1.17.2 ウィンドウ

今説明したワールド座標系は，元々仮空のものであり，我々が実際に見ることのできる座標はあくまでスクリーン座標なのです．

従ってユーザーは，ワールド座標系の中から任意の領域を指定し，スクリーン座標上に表示領域として割り当てなければなりません．N_{88}- BASIC でこの操作をしているのが後述する〝WINDOW″命令であり，この指定されたワールド座標上の領域のことをウィンドウ（のぞきまど）と呼びます．次の図では，ワールド座標系の（−1000, −1000）から（1000, 1000）の領域をウィンドウとしています．

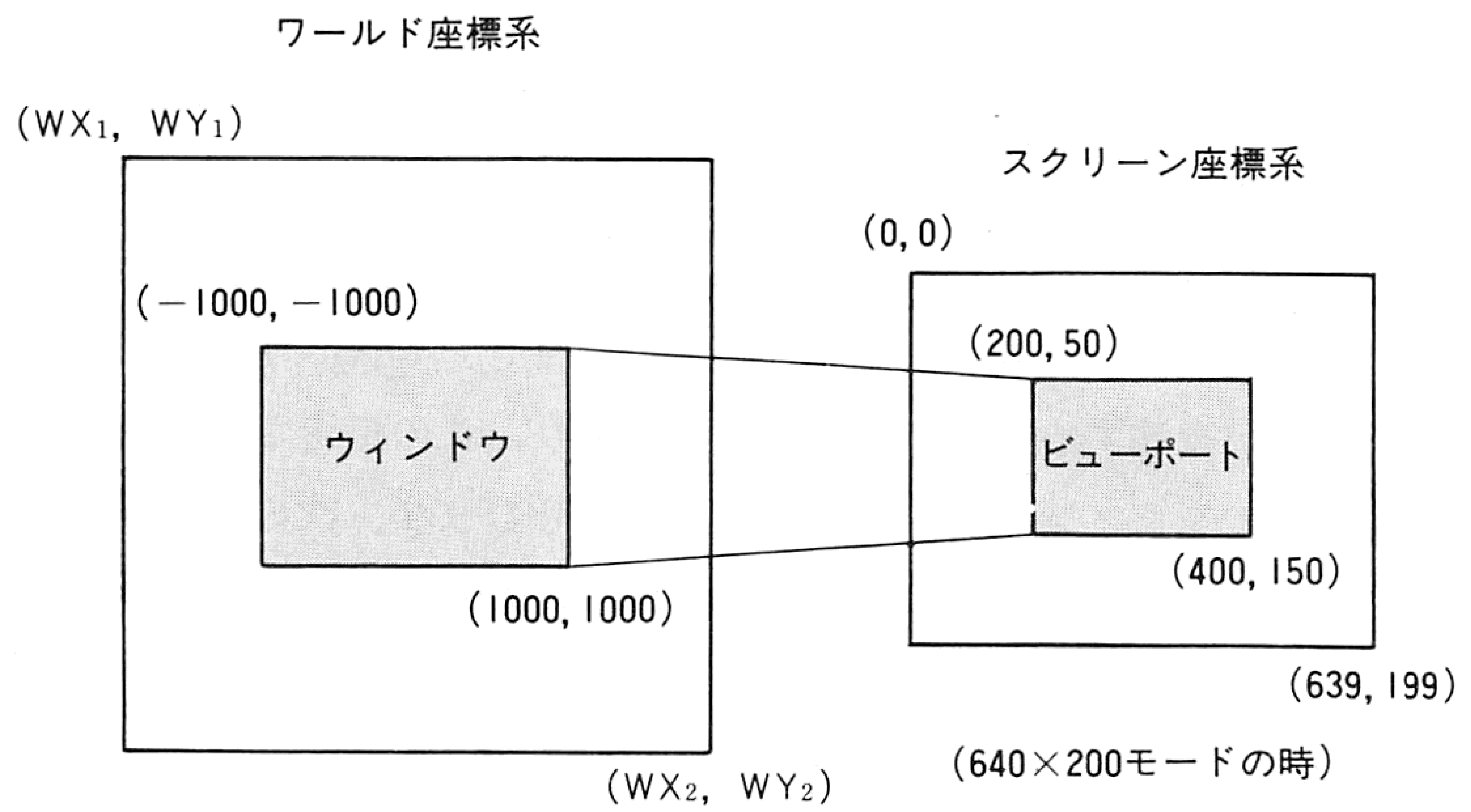


1.17.3 ビューポート

N_{88}- BASIC では，ウィンドウとは別に〝ビューポート″を設定することができます．

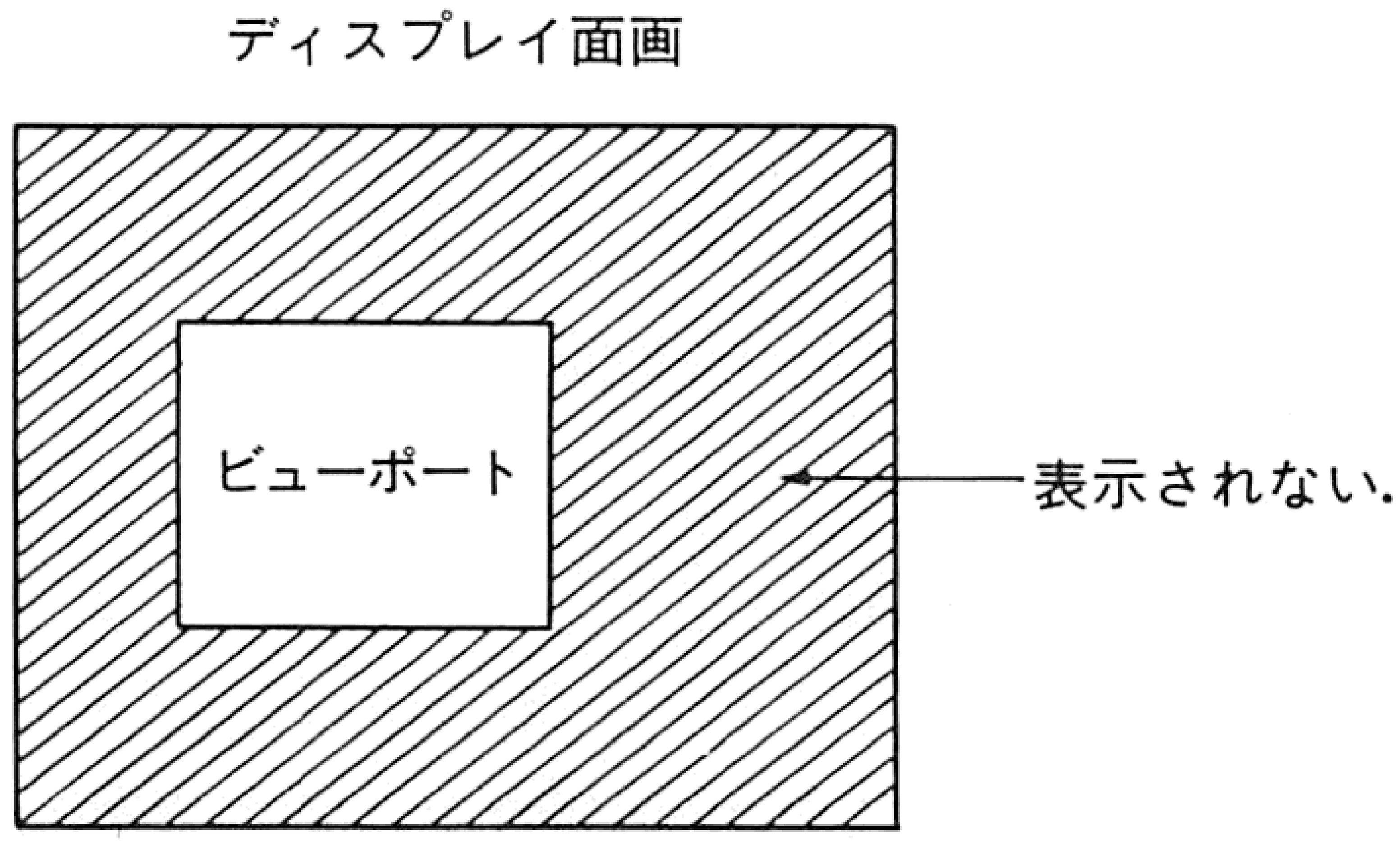
ウィンドウはワールド座標系において，ユーザーがどの領域を表示領域とするかの設定であった訳ですが，このビューポートは，ディスプレイ画面上のどの領域で設定された領域を表示するかの設定です．この操作は，後述する〝VIEW″命令によって行われます．

次の図は，ウィンドウとビューポートの関係です．



この図ではワールド座標系の（−1000, −1000）から（1000, 1000）の領域をウィンドウとし，スクリーン座標系の（200, 50）から（400, 150）という対角線をもつ四角形をビューポートに設定しています．

このようにビューポートとはディスプレイ画面上で見ることのできる領域なのです．一度ビューポートを設定すると，その外側のグラフィック画面には，いっさい表示されることはありません．従ってビューポートを設定するということはモニタのディスプレイ画面の大きさを設定するのと同じと考えることもできます．

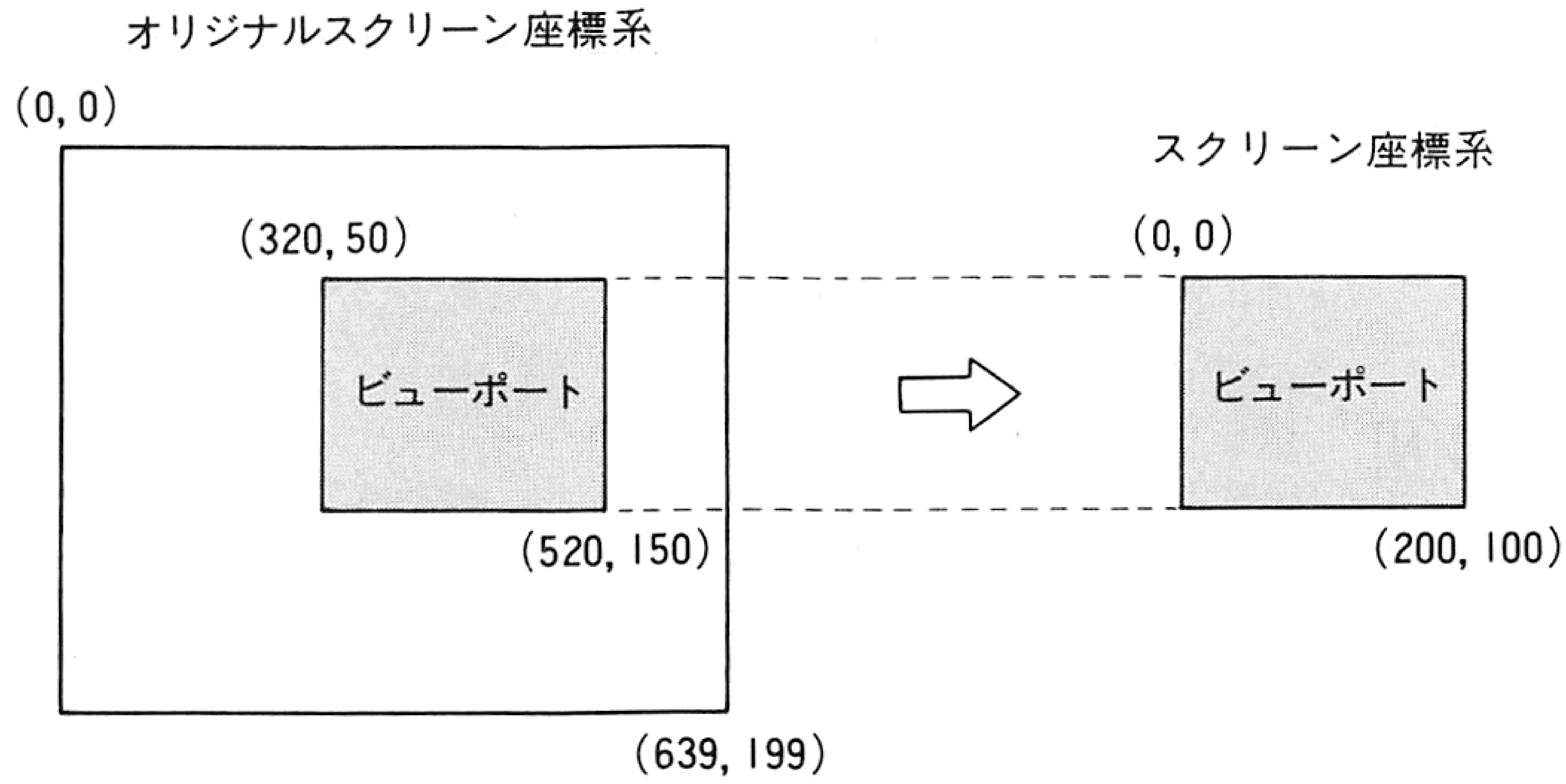


このビューポート内の座標系が先程紹介したスクリーン座標系なのです．このスクリーン座標系とはビューポートの中にのみ存在するもので，この座標系の各点はディスプレイ画面の１ドットに対応しています．これはあくまで物理的な意味の座標系であり，前記したワールド座標系とは異なります．この座標系では，ビューポートの左上の頂点が必ず原点（０，０）に設定され，そこから画面上の各ドットを座標の１点として物理的に展開されます．

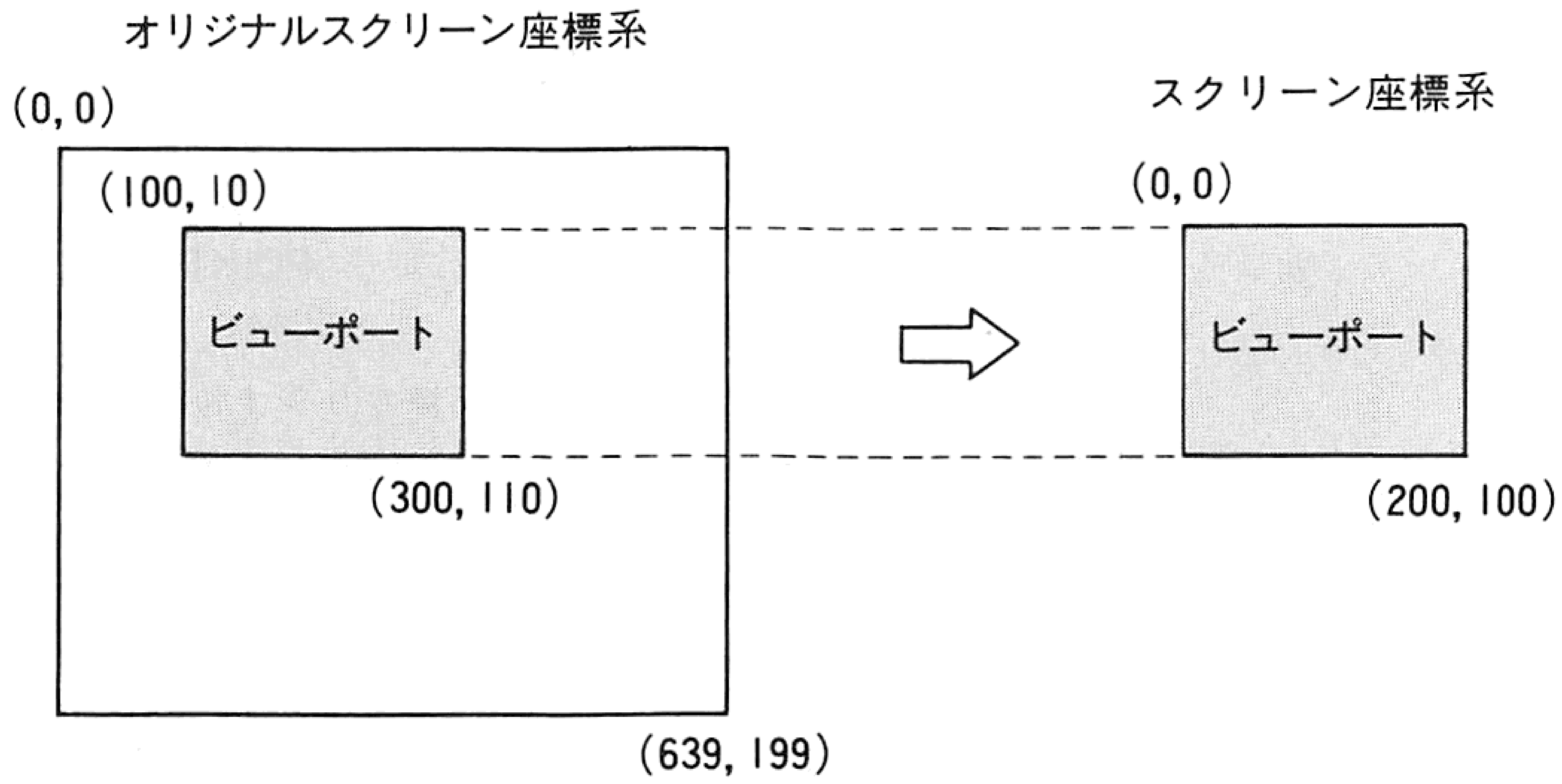
次の図はディスプレイ画面上の異なる位置に同じ大きさのビューポートを設定した例です．この場合，２つのビューポート内のスクリーン座標系は全く等価と

なります.

（1）



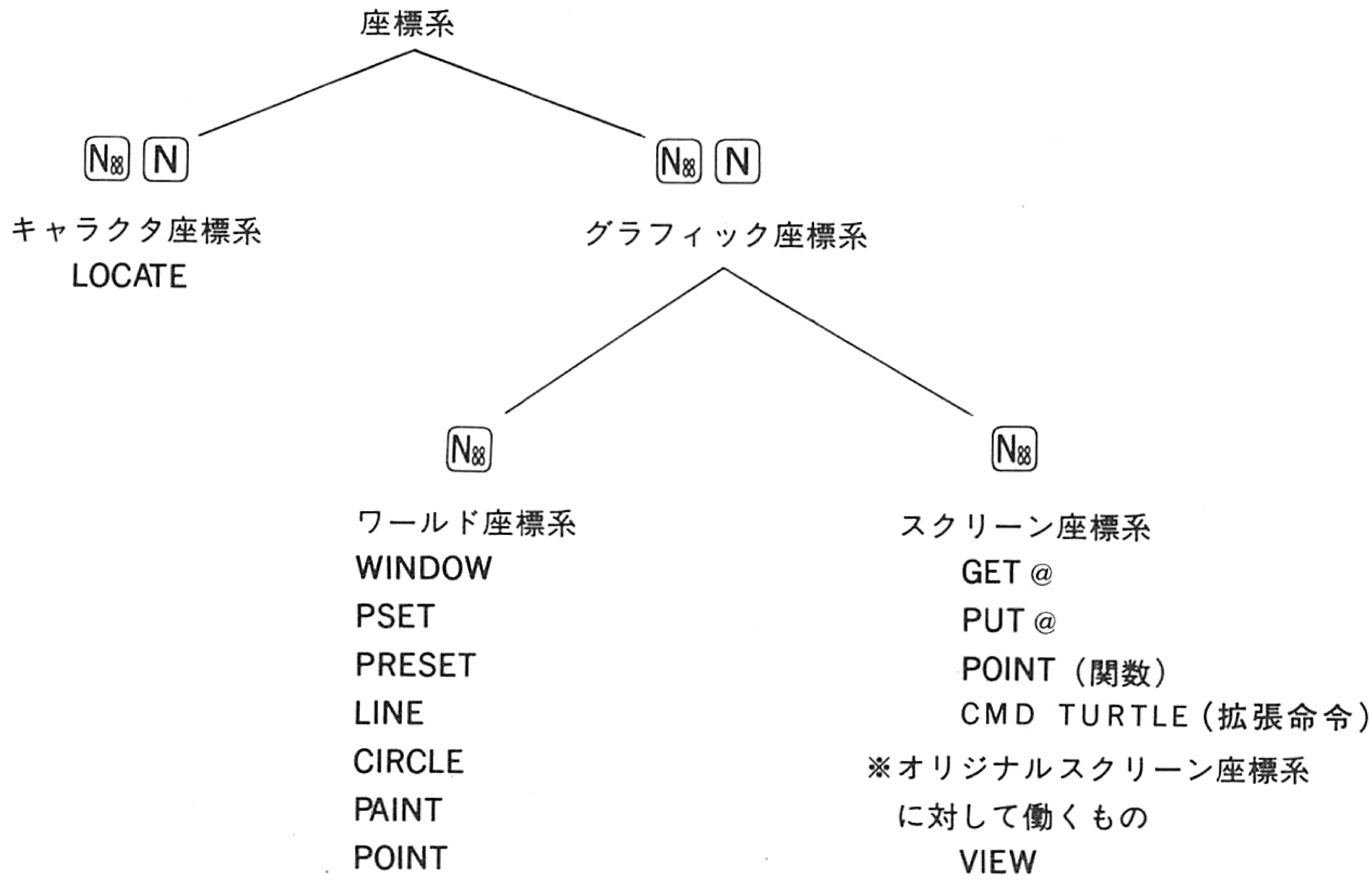
（2）



このビューポート内に展開されるスクリーン座標系の中で，初期状態におけるディスプレイ画面全体をビューポートとした場合の座標系を特にオリジナルスクリーン座標系と呼ぶことにします．従って，オリジナルスクリーン座標とは画面の物理的な座標系ということになります（もし640×200の分解能の画面では，（0，0）－(639，199) の座標系となります).

1.17.4　座標系のまとめ

色々な座標系が出てきましたから，ここで整理しておきましょう．次に示す図はそれぞれの座標系の関係です．また各座標系の下に列挙されているのは，2章，3章，4章で説明する命令のうちで，その座標系に対して働くものです．



キャラクタ座標系　―テキスト画面に存在する座標，座標の1点は1キャラクタに対応する．

グラフィック座標系―グラフィック画面に存在する座標．以下のすべての座標を含んだ意味で使われる概念的なもの．N - BASICの場合は，各ドットに対応した座標として使われる．

ワールド座標系　―BASICが設定した仮空の座標系．ほとんどのグラフィック命令はこの座標系に対して働く．

スクリーン座標系　―ビューポートを設定した時に，ビューポート内に存在する座標．

※オリジナルスクリーン座標系はディスプレイ画面のすべてをビューポートと考えた時のスクリーン座標系．

注）　ちょっと難かしい概念ですが，ウィンドウはワールド座標系に対しての領域指定であり，ビューポートはディスプレイ画面に対しての領域指定であるということと，2章以降で解説するN_{88}- BASICのほとんどのグラフィック命令は，ワールド座標系に対して有効であるということを覚えておけば，

さほど混乱することはないでしょう．

またこのウィンドウとビューポートという概念はN_{88}- BASICでのみ通用するものです．従ってN - BASICには，ワールド座標系やスクリーン座標系というものは存在しません．ただし，2章以降の命令説明の際，便宜上この言葉を使うことがあります．N - BASICではその場合，単にディスプレイ画面上のグラフィック座標を意味しているものと解釈してください．

またN_{88}- BASICでも，ウィンドウとビューポートを全く設定しなければ，グラフィック座標系＝スクリーン座標系＝ワールド座標系と考えることができます．

1.18 座標の指定の仕方

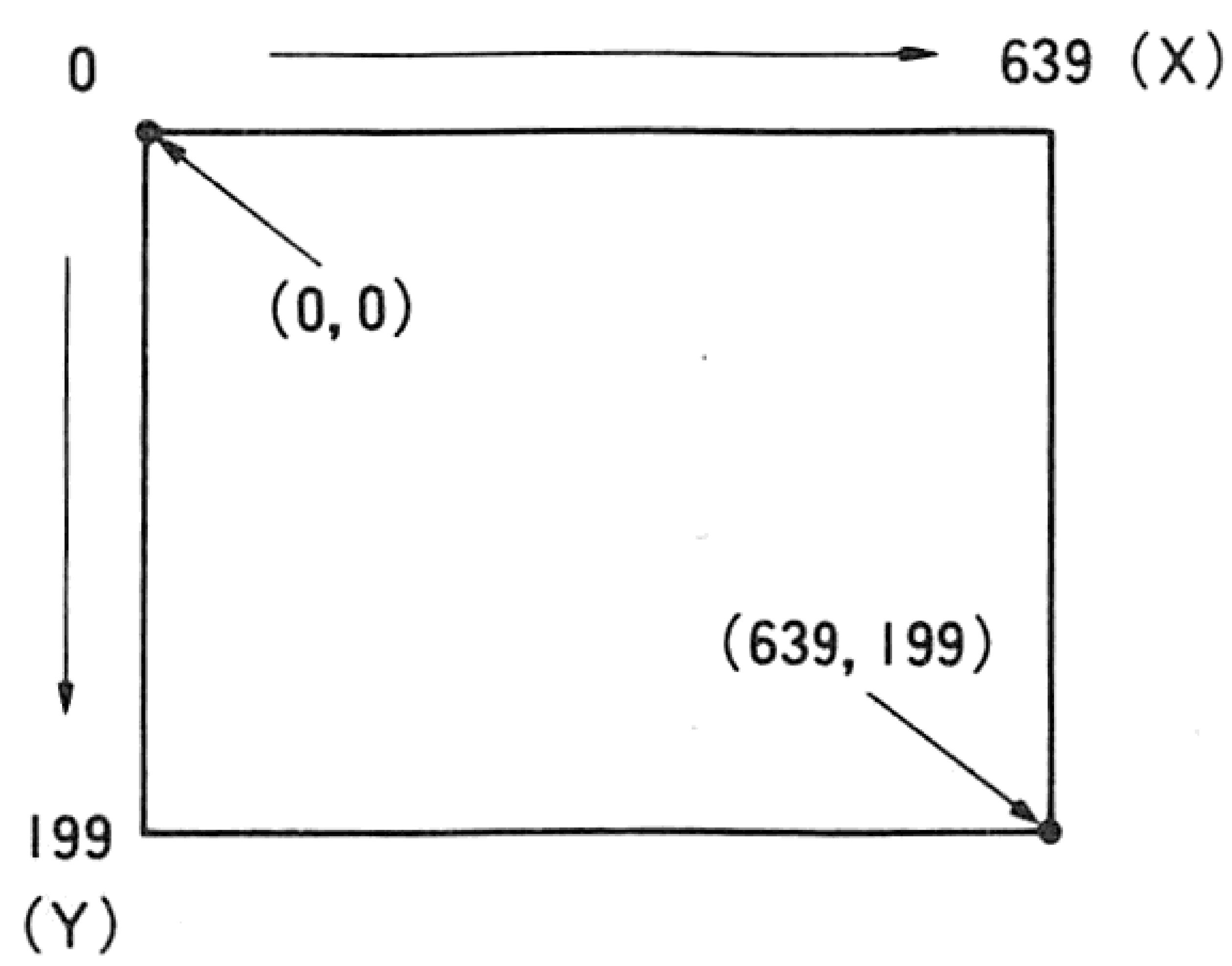
1.16，1.17で色々な座標系を説明しましたが，ここではそれらの座標系に対して種々のグラフィック命令を使う場合の〝座標の指定の仕方〟を説明します．

最も一般的な座標の指定は〝絶対座標〟によるもので，N_{88}(N)- BASICでの座標指定は，普通どの座標系（キャラクタ座標系，スクリーン座標系，ワールド座標系）に対してもこの指定により行います．ただし，N_{88}- BASICのワールド座標系に対しての座標の指定には，相対座標による方法も許されています．

次にこの2つの指定の仕方について説明します．

1.18.1 絶対座標による指定の仕方

絶対座標とは，X座標，Y座標が決まることによって一意的に決まる座標のことです．例えば，次のような座標系があったとすると，左上の頂点は常に（0，0）であり，右下の頂点は常に（639，199）です．〝絶対座標による指定〟とはこの絶対座標を使って位置を表す方法をいいます．



絶対座標

これは最も一般的な指定法で，N_{88}(N)-BASICが持つすべての座標系に対して有効です．2章以降で解説するグラフィック命令などにおいて座標指定する場合，普通はこの方法により行います．例えば，PSET (X, Y) という命令はワールド座標系内の1点にドットをセットする命令ですが，これは次のような書式で書くことができます．

例)　　PSET(100, 100)

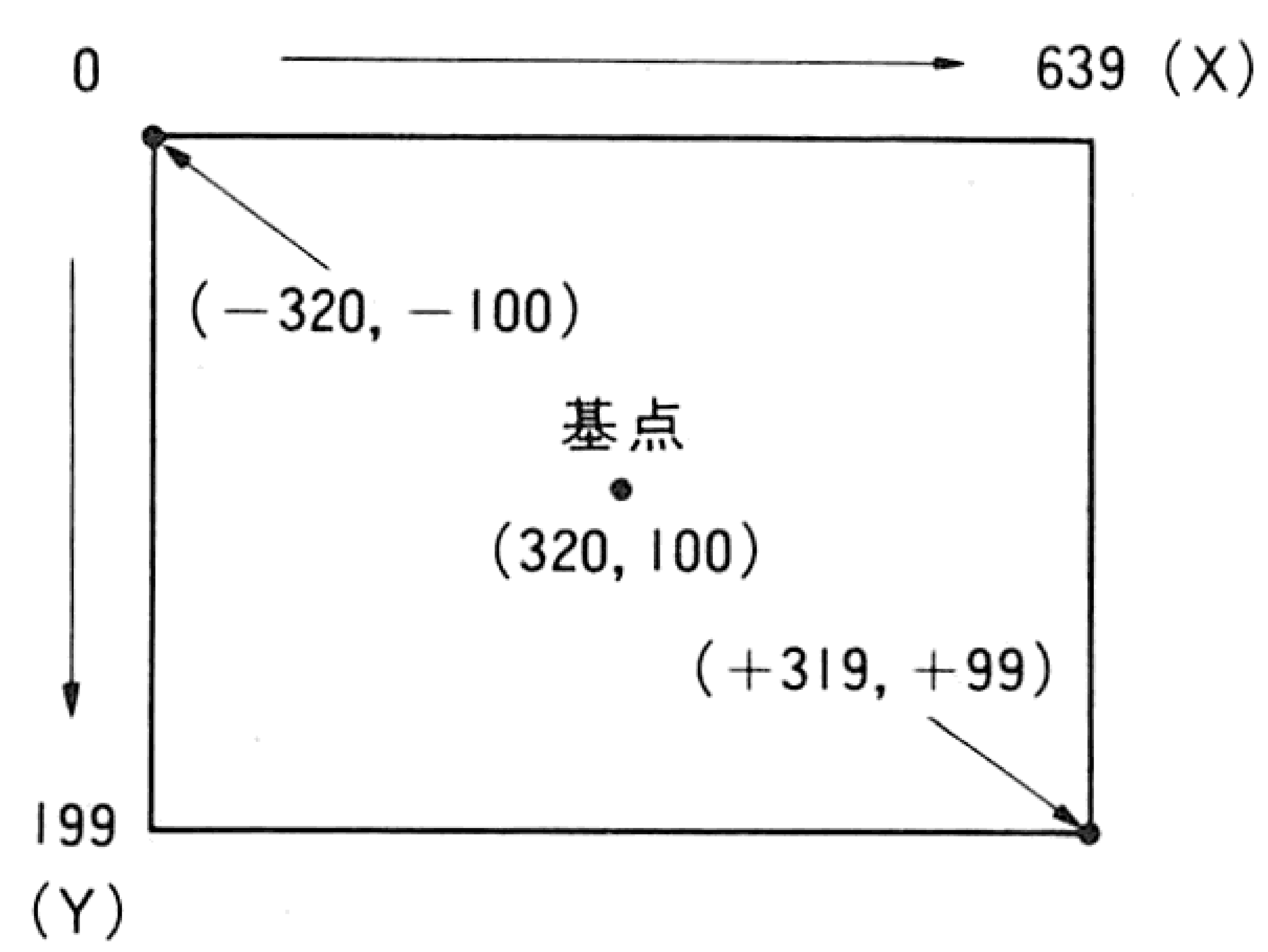
この場合，ワールド座標の (100, 100) にドットがセットされます．その他の命令の場合も絶対座標による指定では，すべてこのように (X, Y) という形になって表されます．

1.18.2 相対座標による指定の仕方 (N_{88}-BASIC)

N_{88}-BASICのワールド座標系に対するグラフィック命令での座標指定には，上記したように〝相対座標による指定〟が許されています．

〝相対座標〟とは，座標系内のある1点を基点とした時，X方向，Y方向にどれだけ相対的に移動するかによって決められる座標です．〝相対座標による指定〟とは，この相対座標を使って位置を表す方法をいいます．

例えば，次のような座標系があり，基点が (320, 100) に設定されていたとすると，左上の頂点は (−320, −100) であり右下の頂点は (+319, +99) となる訳です．



相対座標

この指定法の優れている点は，相対移動量を表す (X, Y) は常に同じ値であっても，基点の位置を動かすことによって違った位置を表現することができることです．

N_{88}-BASICはこの基点に当たるものとして，Last referenced Point (以降〝LP〟と略す) を持っています．このLPは最後に参照された座標という意味で，グラフィック命令で座標を与えた場合，最後に指定された座標を値として持ちま

す.

例えば，PSET(X，Y) という命令が実行された場合，LP は (X，Y) という座標に設定されることになります（2章，3章の POINT 参照).

N_{88}-BASIC のグラフィック命令で相対座標を使って座標指定を行う場合は，次のような書式になります.

例)　PSET　STEP(X，Y)

これはドットをセットする命令ですがその他の場合も同様で，座標を表す (X，Y) の前に〝STEP〟を付けて (X，Y) が相対量であることを表します.

N_{88}-BASIC のグラフィック命令で相対座標の指定が許されているのは次に示す命令です（おのおのの命令の機能については2章で説明します).

PSET	PRESET
LINE	PAINT
CIRCLE	POINT(ステートメント及び関数)
GET@(但し制約あり)	

1.19 カラー

1.19.1 N-BASIC のカラー

N-BASIC のカラーは黒，青，赤，紫，緑，水色，黄，白の8色を使うことができ，それぞれ下記のカラーコードによって指定します.

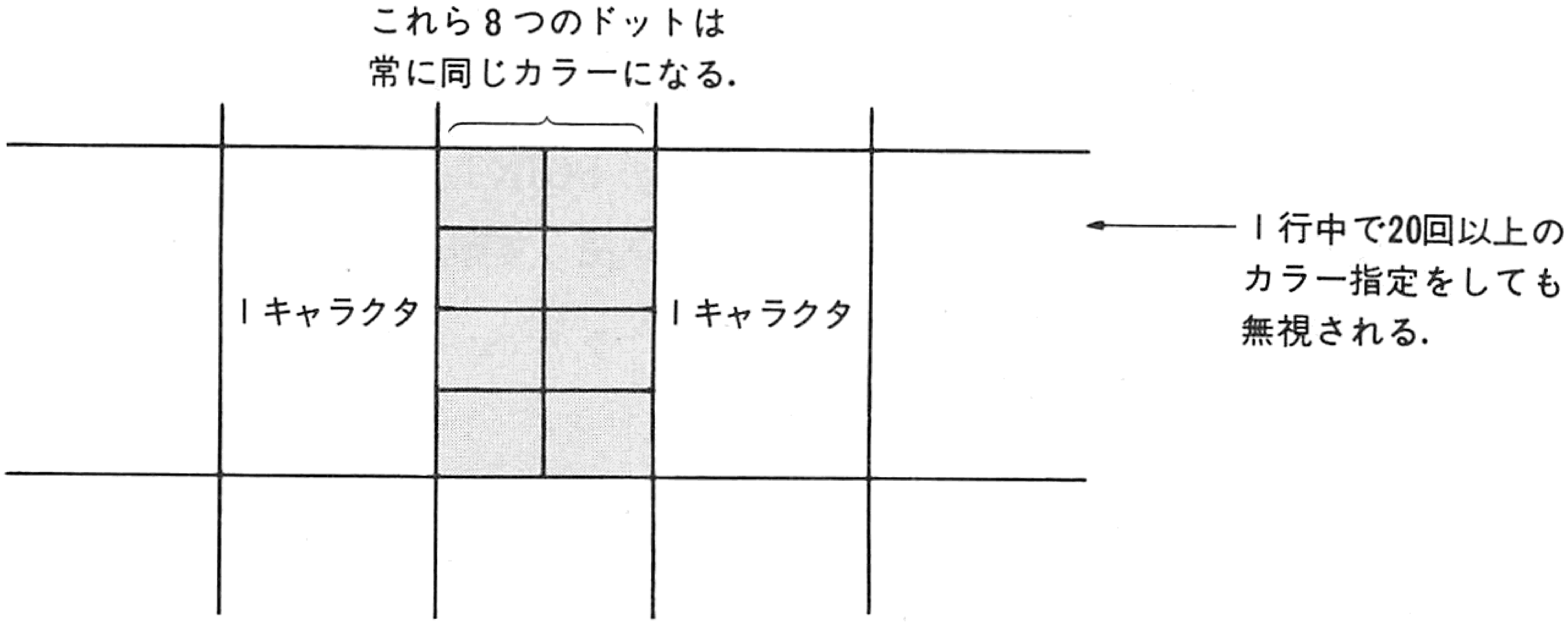
0	1	2	3	4	5	6	7
黒	青	赤	紫	緑	水色	黄色	白

カラーコード

ただし，これらは，〝画面モード〟の所で説明したように，あくまでキャラクタ単位でしか指定することはできません．従って，キャラクタ文字を1文字ずつ塗り分けるということはできますが，ドットグラフィックスの場合1キャラクタ内の隣接する複数のドットを違う色で塗り分けることはできません.

その他，1行中で20回以上のカラー指定を行った場合，それ以降のカラーは無視され，最後に指定したカラーが有効となります.

次の図は以上の説明を示したものです．これらカラーの設定については，COLOR，CONSOLE の項を参照してください.



1.19.2 N_{88}-BASICのカラー

N_{88}-BASICのカラーも8色で，使える色もN-BASICの場合と全く同じです．ただし，N_{88}-BASICはテキスト画面とグラフィック画面を別々に持っていますから，個々にカラー指定を行うことができます．

（1） テキスト画面のカラー

テキスト画面のカラー機能は，N-BASICの場合と全く同様で，その設定もカラーコードにより行います．1.19.1の〝N-BASICのカラー〟を参照してください．

（2） グラフィック画面のカラー

N_{88}-BASICのグラフィック画面でカラー表示が許されているのは，横640×縦200の分解能の時のみです．この状態にある時，ユーザーは各ドットごとに8色のうちから任意の1色を指定することができます．従って，どんなに隣接したドットであっても完全に別の色で塗り分けることが可能です．

ただしテキスト画面のカラー指定は〝カラーコード〟により行いましたが，グラフィック画面のカラー指定は〝カラーパレット〟により行います．次にこのパレットについて説明しましょう．

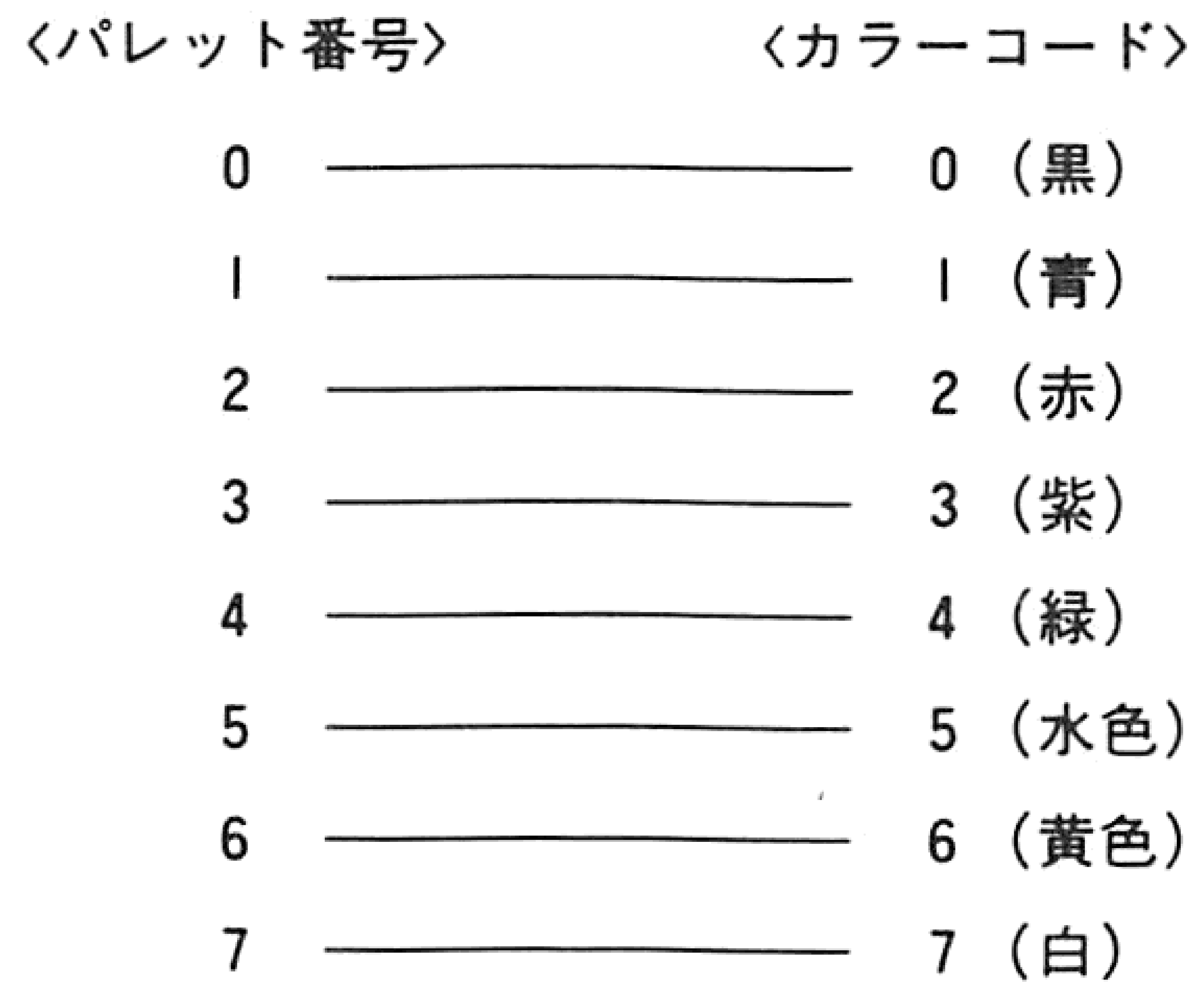
（3） カラーコードとパレット番号

N-BASICでの色の指定はすべて〝カラーコード〟によって行われていました．しかしカラーコードとは，色そのものを表すコード番号ですから（0—黒，1—青，2—赤，…），融通性や汎用性に欠けています．例えば，色指定を1として青にしてしまったものは，絶対的に〝青〟という意味を持ち変更することができません．

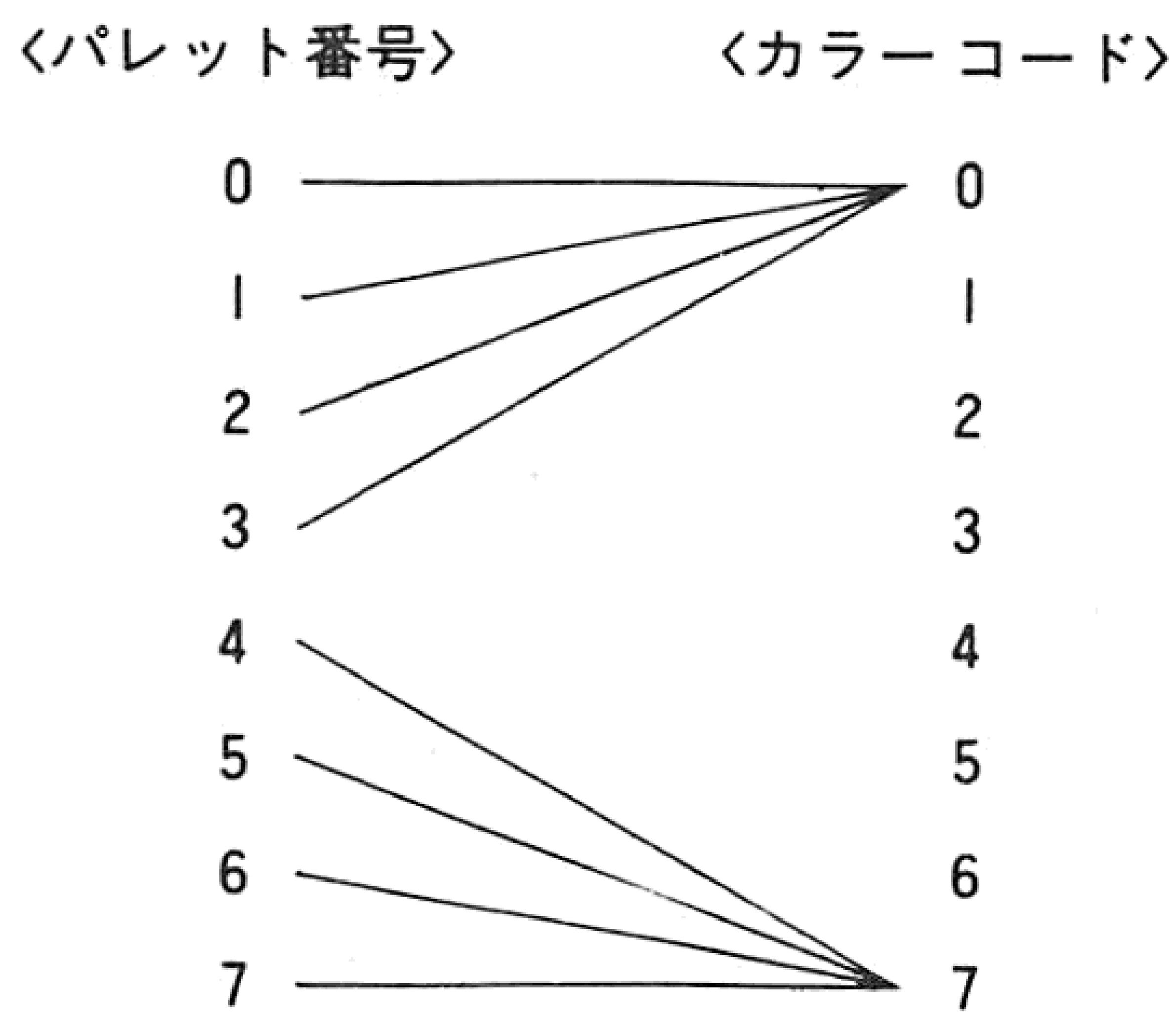
そこでN_{88}-BASICのグラフィック画面に対するカラー操作には，〝パレット〟という新しい概念が導入されました．〝パレット〟とは，その名の示す通り絵を描く時に使うパレットという意味です．N_{88}-BASICでは0番から7番までの8つ

のパレットを持っていて，ユーザーはこれらのパレットに好みの色を設定することができます．

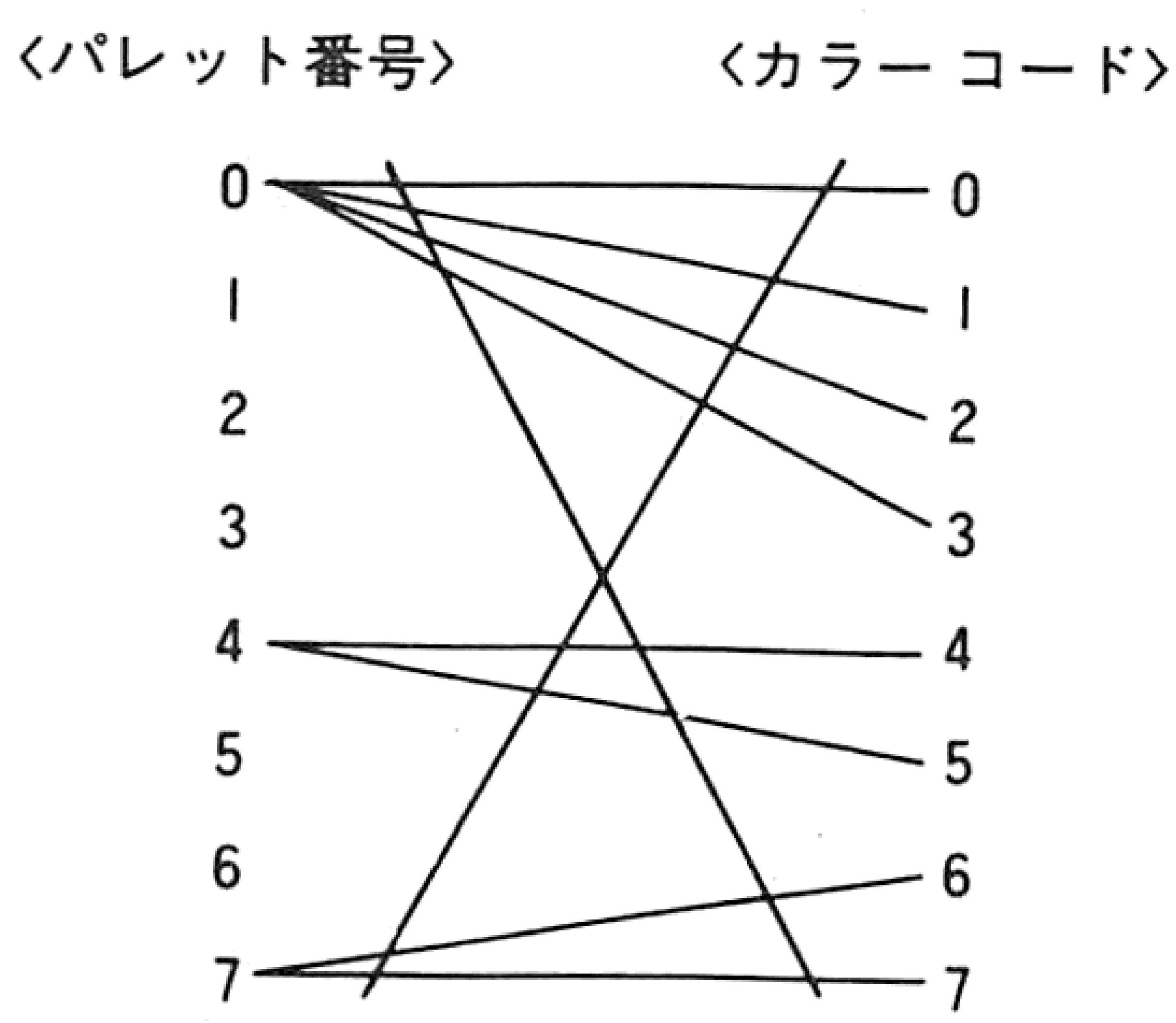
次の図はパレットの設定の一例でパレット0が黒，パレット1が青，パレット2が赤，…というように対応させています（N_{88}-BASICが起動した最初の状態はこのように，〈パレット番号〉=〈カラーコード〉の対応となっている）．



また次のように対応させると，パレットの0～3は黒，4～7は白となり，画面に表示される色は白と黒の2色ということになります．



ただし，次のように1つのパレットに複数の色を対応させて混合することは許されていません．



このようにしてパレットが設定されると，その瞬間から，グラフィック画面に表示されている色及びこれからグラフィック画面に描こうとする色はすべて変更されたパレットの影響を受けます．例えば，最初パレット番号1に赤が設定されている状態で画面に線を描いたとします．次にそのパレット番号に青を設定したとすると，画面に描いてあった赤の線は瞬時に青の線になってしまうのです．

このようにパレットには，決った色という概念はなくいつでも色を切り替えることができますから，プログラムに融通性があり多彩な色の変化を実現することが可能です．

後述する N_{88}- BASICのグラフィック命令 (COLOR, PSET, LINEなど) の色指定はすべてこのパレット指定により行いますから，カラーコードとパレットの意味を正しく把握して御使用ください．

これら〝グラフィック画面〟，〝カラーコード〟，〝パレット〟は，絵を描く時の〝カンバス〟，〝絵の具〟，〝パレット〟に例えて考えれば理解しやすいことでしょう．

1.20 割り込み (N_{88}-BASIC)

コンピュータの頭脳であるCPU (Central Processing Unit) は，原則として同時に2つ以上の処理をすることができません．従って，1つの仕事を処理している間は，他で何が起ころうとCPUは，その処理に移ることはできませんし，それを知ることさえできないのです．この事は，1つの決った一連の流れを頭から処理するようなプログラムの場合，さほど問題にはなりませんが，1つのプログラムの処理中に，即実行しなければならないような特別な事象が起きた時に，その処理を先に実行するようなことを行わせるようなプログラムの場合問題です．

〝割り込み〟とは，このような処理を実現するための手段です．例えば，CPUがある処理を実行している時，特別な事象が発生したとします（ライトペンが押された，ファンクションキーが押されたなど)．このような場合，専用のハードウェアはCPUにその事象が起きたことを，ハードウェア的手段で伝えます．そこでCPUは，現在実行中の処理を一時停止し，特別な事象の処理を優先的に実行し，その処理の終了後，元の処理に復帰するのです．

以上のような流れを〝割り込み〟と呼ぶ訳ですが，PC-8801MKIIは，多くの周辺装置でこの割り込みを使えるようになっていますし，N_{88}-BASICもこれらすべての割り込みをサポートしています。ただし，N-BASICでは，これらの割り込み操作はサポートされていません．

次に，N_{88}- BASICがサポートしている割り込み機能を示します．

ファンクションキー	(ON KEY GOSUB 参照)
ストップキー	(ON STOP GOSUB 参照)
ヘルプキー	(ON HELP GOSUB 参照)
RS - 232C	(ON COM GOSUB 参照)
ライトペン	(ON PEN GOSUB 参照)
リアルタイム・タイマ	(ON TIME$ GOSUB 参照)

1.21 ファイル

ファイルとは意味を持つ情報の集まりで，N_{88}(N) - BASIC は外部機器との入出力を行う時，このファイルを一つの単位として管理します．このファイルにはデータファイルとプログラムファイルの 2 つがあります．

さらに N_{88}(N) - BASIC ではファイルを使用するにあたって，そのファイルにファイル番号（チャネル番号）を割り当て，ファイル番号でファイルの入出力を行うため能率よくファイル操作を行うことができます．

1.22 ファイルディスクリプタ（N_{88}-BASIC）

N_{88} - BASIC では，入出力機器一般についてもファイルの概念を導入することによってすべての入出力機器との入出力操作を統一しています．これらは〝ファイルディスクリプタ″によって区別されます．ただし，N - BASIC にはこの概念はありません．

1.22.1 ファイル番号とチャネル番号

N_{88}- BASIC では入出力操作はファイルやチャネルの指定によって実行します．ファイル番号とチャネル番号は同一の概念で，入出力のためのメモリ領域(バッファ）に対して付けた固有の番号です．

ファイル番号は補助記憶装置に対して用い，チャネル番号は入出力（通信）機器に対して用います．ファイル番号とチャネル番号は同一の概念ですから重複して指定することは許されません．N_{88}- BASIC が起動した時指定する数は，このファイル（チャネル）の数で，この数の範囲内であれば何個でもファイル番号を指定することができます．

1.22.2 ファイルディスクリプタの構成

ファイルディスクリプタは前記したように各ファイル（デバイスも含む）を区

別する名称で，次のような構成の文字列で表されます．

"〈デバイス名〉:〔〈オプション〉〕〔〈ファイル名〉〕"

ファイルディスクリプタは必ずダブルクォーツ(")で囲まなければなりません．デバイス名，ファイル名は場合によって省略することができます．デバイス名が省略できるのは，N_{88}- BASIC (ROMバージョン) の場合カセットテープ1 ("CAS1:")，N_{88}DISK-BASIC (ディスクバージョン) の場合フロッピーディスク1 ("1:")を指定する時です．

1.22.3 デバイス名

〈デバイス名〉は入出力機器の名称を表すもので，アルファベット4文字もしくはアルファベット3文字と数字1文字に"：" (コロン) を付けたものを原則としています．ただし，フロッピーディスク装置については数字 (ドライブ番号を示す) 1文字と "：" の簡略形が用いられます．現在 N_{88}- BASIC に定義されているデバイス名には次のものがあります．但し，ディスク(n:)，SCRN :，KYBD : は N_{88}DISK-BASIC でのみ供給されます．

デバイス名	機械名称	入力	出力	備考
LPT1： またはLPT:	プリンタ	×	○	
SCRN：	スクリーン	×	○	N_{88}DISK-BASICで使用可
COM1： またはCOM:	RS-232Cポート 1	○	○	
CAS1： またはCAS:	カセットテープ 1	○	○	N_{88}-BASICでのデフォルト，転送速度1200ボー
CAS2：	〃 2	○	○	転送速度600ボー
1：	フロッピーディスク 1	○	○	N_{88}DISK-BASICでのデフォルト
2：	〃 2	○	○	
3：	〃 3	○	○	
4：	〃 4	○	○	
5：	〃 5	○	○	
6：	〃 6	○	○	
7：	〃 7	○	○	
8：	〃 8	○	○	
KYBD：	キーボード	○	×	N_{88}DISK-BASICで使用可

1.22.4 オプション

＜オプション＞は，コミュニケーション（RS－232C）をデバイスとして指定した時のみ意味を持つもので，パリティチェックの有無など種々のモードを設定するのに用いられます．詳しくは2章TERM及びユーザーズ・マニュアルを参照してください．

1.22.5 ファイル名

〈ファイル名〉とは1.21で説明したプログラムファイルやデータファイルに付ける名前で，これはユーザーが指定します．ファイル名を必要とするのは，カセット，フロッピーディスクなど補助記憶装置との入出力を行う場合で，その他の場合は省略することができます．ファイル名は次のような書式をとります．

〈ファイル名〉〔．〕〔〈拡張ファイル名〉〕

最初の〈ファイル名〉は6文字，ピリオドに続く〈拡張ファイル名〉は，もしあれば3文字までの文字列より構成されます．もしそれぞれの文字数がそれ以下の場合には空いた部分に空白が埋められます．一般に〈ファイル名〉がファイルの名前を，〈拡張ファイル名〉がファイルの性質を表すようにしますが，ユーザーによって自由に使うことができます．通常ファイル名と言う場合は〈拡張ファイル名〉まで含んだものを意味します．

ただし，ファイル名の中に，コロン（：）及びキャラクタコード0と255で表される文字を含むことは許されません．

また最初の〈ファイル名〉が6文字を越えて指定された場合，先頭から6文字が〈ファイル名〉と，続く9文字までの3文字が〈拡張ファイル名〉と見なされ，それ以降の文字は無視されます．

1.23 ラベル（N88-BASIC）

一般にBASIC言語においては，プログラムの流れを変える命令として〝GOTO〟，〝IF　THEN〟などが使われますが，これまでのBASICではこれらの飛び先の指定をすべて行番号で行っていました．次のプログラムはその一例です．

```
10  INPUT A
20  IF A<0 THEN 80
30  IF A>0 THEN 60
40  PRINT "zero"
50  GOTO 90
60  PRINT "plus"
70  GOTO 90
80  PRINT "minus"
90  END
```

これは，10行で入力された値が正か負かゼロかを調べるプログラムです．ここで処理の流れを変える命令が20，30，50，70行に使われていて，その飛び先の指定はすべて行番号になっています．しかし，この行番号はあくまで数字の並びであり，我々にはなじみにくく，プログラムの作成，デバッグ時においてかなり混乱を招きます．

そこで N_{88}-BASIC では，飛び先の指定にラベル名の使用が許されました．ラベルとは，どのようなものかを説明する前にまずそのラベル名を使って先程のプログラムを書き変えてみましょう．

```
10  INPUT A
20  IF A<0 THEN *MINUS
30  IF A>0 THEN *PLUS
40  PRINT "zero"
50  GOTO *EXIT
60  *PLUS : PRINT "plus"
70  GOTO *EXIT
80  *MINUS : PRINT "minus"
90  *EXIT : END
```

このようにラベル名とは，ユーザーが分岐先の目印として独自に付けることのできる名前なのです．

このラベル名の使用においては，次の注意を守らなければなりません．

1. ラベル名の頭には必ずアスタリスク（*）を付けなければなりません．
2. 先頭のアスタリスクを除いて，ラベル名は必ず英文字で始まらなければなりません．
3. ラベル名に使用できる文字は，先頭のアスタリスクを除いて英文字と数字とピリオド（.）であり，大文字と小文字の区別はありません．

4. BASICの予約語(PRINTなど)をラベルとして使用することはできません. ただし, 中に含む場合は構いません.
5. ラベル名の長さは, プログラムの1行の許可範囲(1行255文字)によってのみ制限を受けます.
6. 参照されるラベル名(呼ばれる方のラベル名)は, 必ず行の最初になければなりません.
7. 1行中でラベル名の後に続けて命令を記述し, マルチステートメントとする時は, コロン(:)またはスペースによって区切ります.

 以上のような制限を無視すると, プログラムのロード及び実行直後に〝? SN Error〞が表示されます. (行番号は表示されません。) この場合, LISTコマンドやEDITコマンドは実行できますから, ラベル名を訂正してください.

この他参照されるラベル名に同じものがあった場合, 2重定義されているという意味の〝duplicate label〞(DL Error) エラーが生じます. これらのエラーの検出は, 他のエラーメッセージの場合と異なり, 〝RUN〞した時プログラムの実行に先だって行われますから, ラベル名のエラーがあるとプログラムを1ステップも実行することができません.

また, この場合のエラーメッセージは, エラー箇所を示す〈行番号〉は伴いません.

ラベル名は, プログラム中で分岐の目印として使用する他, コマンドレベルで〝LIST〞, 〝DELETE〞など〈行番号〉を対象とするパラメータにはすべて使用することが可能です.

例)
```
LIST *START-*LAST
DELETE *LOOP1-*EXIT1
```

1.24 エラーメッセージ

N_{88}(N)-BASICは, プログラムの実行を中断させなければならないようなエラーを実行時に検出した時, エラーメッセージをプリントしコマンドレベルに戻ります.

ダイレクトモードでのエラーメッセージの書式は,

XX

プログラムモードでの書式は，

XX in IIIII

XXはエラーメッセージで，11111はエラーが検出された行番号です．

N88（N）-BASICのエラーコードとエラーメッセージの内容については，APPENDIXの〝エラーコード表〟を参照してください．

ただしN88-BASICのエラーメッセージはROM BASICとDISK BASICで異っており，ROM BASICでは簡略した形で出力されます．このマニュアルにおいてフルセンテンスのメッセージの後にカッコで示されているのがこれにあたります．

1.25 拡張命令パッケージ

PC-8801MKIIでは，N88-BASIC用に，拡張命令パッケージが用意されています．

これは，タートルグラフィックス機能や，サウンド機能をサポートするものです．

詳細は，「PC-8801MKIIユーザーズマニュアル」を参照してください．

2章・3章・4章の見方

2章，3章，4章でN_{88}(N)-BASICのすべての命令を解説します．これら3つの章を読む場合，次のことに注意してください．

機能 命令の機能を簡単に示します．

書式 命令の記述の仕方を示します．実際の入力時には次のような決まりに従ってください．

1. アルファベットの大文字で示された項目は，そのまま入力します．入力する場合は，小文字でも大文字でもかまいません．ただし，引用符（"）で囲まれた文字列（ファイル名など）は，大文字と小文字を区別してください．
2. カギカッコ〝〈〟　〝〉〟で囲まれた項目は，ユーザーが指定します．
3. 角カッコ〝〔〟　〝〕〟で囲まれた項目は，オプションであり省略することができます．省略した場合，デフォルト値(BASICによって設定される値)または以前に指定した値が適用されます．
4. 上記のカギカッコ，角カッコ以外の記号でカッコ（ ），コンマ（,），セミコロン（；），ハイフン（-），等号（=）などの記号は示された位置に正しく入力します．
5. 省略記号〝・・・〟の続く項目は，1行の許す長さ（255文字）の内で任意の回数繰り返すことができます．

   例：　〈定数〉〔,〈定数〉・・・〕 の場合　0，10，15

6. 座標指定の所で(Wx，Wy)と記されているのはワールド座標を，(Sx，Sy)はスクリーン座標を表しています．その他単に(X，Y)と記されているのはキャラクタ座標を表しています．またSTEP (x，y) という記法は相対座標による指定ができることを意味しています．ただし，N-BASICグラフィック画面ではワールド座標，スクリーン座標はありませんからすべてグラフィック座標と解釈してください．また相対座標による指定はできません．
7. 書式の中で〈行番号〉またはそれに類して行番号の指定を促しているものは，行番号そのものの他にラベル名を含んでいると解釈してください．ただし，N-BASICでは行番号だけです．

文例 実際の入力の仕方の見本として簡単な例を示します．

解説 命令の使用方法や詳しい機能とそれに関して注意しなければならない点などを説明します．

サンプル プログラム　いくつかの文を交えたプログラム例とその実行結果を示します.

これらの説明は，原則としてN_{88}，N - BASICと区別せず一括して解説しますが，どちらかに固有の場合または特別な事柄の場合，次のように区別します.

N_{88} ―――――― N_{88}- BASIC固有の命令を表します.

N ―――――― N - BASIC固有の命令を表します.

(ディスク) ―― ディスク板BASIC固有の命令を表します.
（N_{88}DISK-BASICまたはDISK-BASICが供給された時に使用可能となります.）

その他区別をはっきりつけるために，N - BASIC固有の事柄については，黒のアミ（　　）をかけて分かりやすくしてあります.

3章〝関数〞の算術関数の中で，初等関数（sin, cos, tanなど）は引数の値として倍精度型の数値を与えると倍精度型で，その他の数値を与えると単精度型でその値を返します．但し，倍精度関数機能はN_{88}DISK-BASICでのみ使うことができます.

またN-BASICでは，ROM版，DISK版共に倍精度関数機能はありません.

第2章

コマンド・ステートメント

AUTO　N88　N

機　能	行番号を自動的に発生する．
書　式	AUTO〔〈行番号〉〕〔，〈増分〉〕
文　例	AUTO　100，5

解　説　AUTOコマンドは，指定された〈行番号〉を発生し，行の入力待ちになります．以後，リターンキーの入力ごとに，〈増分〉で指定された増分ずつ増加した行番号を自動的に発生します．〈行番号〉，〈増分〉ともに省略された場合には，10が省略値として選択されますが，コンマがあって〈増分〉が省略された場合には，直前に実行したAUTOコマンドの増分が用いられます．

CTRL - Cまたは，ストップキーを押すと，AUTOコマンドはその動作を中止し，BASICのコマンドレベルに戻ります．この時，最後に発生された行番号の行は格納されません．

プログラム中に既に存在する行と同じ行番号が発生された場合には，行番号の直後にアスタリスク（*）が表示され，注意を促します．この時，文字を入力してリターンキーを押すと，その行は新しい内容に入れ代わります．また，何も入力しないでリターンキーを押した場合には，その行は削除されます．

AUTOのモード中においても，カーソルエディッテング機能を利用することができます．カーソルを移動して既に入力した行，あるいは表示されていた行のところで，リターンキーを入力した場合には，その行の行番号に増分を足した行番号が次に発生される行番号となります．

N - BASICでは，既に入力されている行であることを示すアスタリスクは表示されません．また，カーソルを移動して，既に表示されている行を修正することはできません．

サンプルプログラム

```
auto 100,20
100 rem -- test --
120 print "auto test"
140 end
160
list
100 REM -- test --
120 PRINT "auto test"
140 END
Ok
```

BEEP　Ｎ88　Ｎ

B

機　能	内蔵スピーカを鳴らす.
書　式	BEEP〔<スイッチ>〕
文　例	BEEP

解　説　内蔵スピーカから音を出したり，止めたりします．<スイッチ>の値が1の場合はON（鳴りっぱなし），0でOFFとなります．

また，<スイッチ>を省略した場合には，PRINT　CHR$(7)を実行したのと同じで，一定時間スピーカを鳴らします．

参照：コントロールコード表

サンプルプログラム

```
100 '-- bell --
110 FOR I=1 TO 20
120  BEEP 1
130  BEEP 0
140 NEXT I
150 BEEP
160 '-- wait routine --
170 FOR J=0 TO 200
180 NEXT J
190 '
200 PRINT CHR$(7)
210 END
```

BLOAD （ディスク） N88

機能	機械語プログラムをロードする.
書式	BLOAD <ファイルディスクリプタ>〔,<ロードアドレス>〕〔,R〕
文例	BLOAD "DEMO.BIN", R

解説 <ファイルディスクリプタ>によって指定された，機械語プログラムファイルをメモリ上にロードします.

ロードされる領域は，BSAVEによりファイルをセーブする際に指定したものと同じですが，<ロードアドレス>が指定された場合には，セーブの際に指定された開始番地ではなく，<ロードアドレス>で与えられた番地からロードし始めます．したがって，<ロードアドレス>を付けてロードするプログラムは，セーブされた際と異なる番地にロードされても，実行可能な性質（リロケータブル）を持っていなければなりません。<ロードアドレス>には変数を使うことができますが，Rで始まる変数を使うことはできません。

Rオプションを指定すると，プログラムをロード後，セーブの際に指定された開始番地から，プログラムの実行を直ちに開始します．この時，既に開かれているファイルはその状態を保持します.<ロードアドレス>が指定されている場合は，実行開始番地も<ロードアドレス>の番地になります.

但し，&H8000～&H83FF及び&HE600～&HFFFFの領域にBLOADすることはできません。

BLOADを実行する前には，データを転送する領域をCLEAR文で確保してください．CLEAR文を実行しない場合は正常に動作しません.

注意：カセットを対象としたBLOAD命令は，実行できません.

参照：BSAVE

サンプルプログラム

```
bload "1:test",&h1000
bload "2:sample",&he000
```

BSAVE （ディスク） N88

機　能	**機械語プログラムをセーブする.**
書　式	BSAVE 〈ファイルディスクリプタ〉，〈開始番地〉，〈長さ〉
文　例	BSAVE "DEMO. BIN"，&HE000，&H2FF

解　説　メモリ上に置かれている機械語プログラムを,〈ファイルディスクリプタ〉で指定されたファイルにセーブします．メモリ上の〈開始番地〉以後の連続する〈長さ〉バイトの内容が，機械語プログラムとしてセーブされます．

セーブされる機械語プログラムは〈開始番地〉から実行できるようにしておくと，BLOAD により，プログラムのロードと実行が行えます．

但し，&H8000～&H83FF の領域を BSAVE することはできません．

注意：カセットを対象とした BSAVE 命令は，実行できません．

参照：BLOAD

サンプルプログラム

```
bsave "1:test",&he300,&h35
bsave "2:sample",&hd000,&h1000
```

CALL（ディスク） N88

機　能	**機械語サブルーチンを呼び出す.**
書　式	CALL 〈変数名〉〔(〈引数〉〔, 〈引数〉…〕)〕
文　例	CALL MSUBR (ARG1, ARG2)

解　説　メモリ中に用意された機械語サブルーチンに制御を移します.

〈変数名〉は，機械語サブルーチンの実行開始番地を指定し，〈変数名〉で指定された変数に代入されている値が，実行開始番地となります．ここで用いる変数には配列変数を用いることはできません.

〈引数〉は，機械語サブルーチンに渡す変数を指定します．引数としてはすべての型の変数を指定することができますが，定数や式を渡すことはできません.

CALL によって呼び出されるサブルーチンは，機械語の RET 命令により，BASIC に戻すことができます.

機械語プログラムの作り方や引数の渡し方については，APPENDIX を参照してください.

参照：USR，APPENDIX・機械語プログラムの作り方

C

サンプルプログラム

```
100 CLEAR ,&HDFFF
110 BLOAD "sample"
120 CULC=&HE000
130 WORK=&HE100
140 ARG1%=10
150 ARG2%=23
160 CALL CULC(ARG1%,ARG2%)
170 PRINT PEEK(WORK)
180 END
```

CHAIN（ディスク） N88

機能　プログラムを連結し実行する．

書式　CHAIN〔MERGE〕〈ファイルディスクリプタ〉〔，〈行番号〉〕
〔，ALL〕〔，DELETE〈範囲〉〕

文例　CHAIN MERGE "1 : TEST", 500, ALL, DELETE 500 - 600

解説　〈ファイルディスクリプタ〉で指定されたプログラムをロードし，実行します．

メモリ上にあるプログラム中で，既に開かれているファイルはその状態を保持します．〈行番号〉は呼び出されたプログラムの実行開始行を指定するもので，省略した場合はプログラムの最初から実行します．ここで指定する行番号は，RENUM を実行しても書き換えられません．またラベル名も使うことはできません．

ALL オプションは呼び出されたプログラムに変数，配列を引き渡すのに使います．これを指定した場合すべての変数，配列が引き渡され，省略した場合いっさい引き渡されません．必要な変数，配列だけを引き渡したいという場合は COMMON 文を使います（COMMON 参照）．

MERGE オプションは，現在メモリ上にあるプログラムと指定されたプログラムとを混合（マージ）し，結果のプログラムを指定した〈行番号〉から実行します．〈行番号〉が省略されている場合は，混合したプログラムの最初から実行します．ただし，MERGE オプションを指定した場合，呼び出されるプログラムは必ずアスキーセーブ（SAVE 参照）されたものでなければなりません．

DELETE オプションは MERGE オプションを指定する時のみ意味をもつもので，プログラムを混合する前にあらかじめ，プログラムの指定した範囲を削除しておきます．ここで指定する行番号の範囲は，RENUM を実行することによって書き換えられます．またこの行番号の代わりにラベル名を使うこともできます．

注意：MERGE オプションと ALL オプションを両方とも指定しない場合は，DEFINT，DEFSNG，DEFDBL，BEFSTR，DEF FN，OPTION BASE，ON ERROR GOTO 等の宣言文は無効になります．また ALL オプションだけを指定しても，DEF FN，OPTION BASE，ON ERROR GOTO 等は無効になります．

参照：COMMON，MERGE，RENUM

サンプル
プログラム

```
100 '-- main program --
110 PRINT "< main program >"
120 DIM A%(10)
130 FOR I=0 TO 10:A%(I)=I:NEXT I
140 A$="N88-BASIC"
150 A%=12:B=1.23:C#=.000000012345678#
160 '
170 PRINT A$
180 PRINT "A =";A%,"B =";B,"C =";C#
190 FOR I=0 TO 10:PRINT A%(I);
200 NEXT I:PRINT
210 '
220 CHAIN MERGE "1:test",300,ALL

run
< main program >
N88-BASIC
A = 12          B = 1.23        C = .000000012345678
 0  1  2  3  4  5  6  7  8  9  10
< chained program >
N88-BASIC
A = 24          B = 1.5129      C = 1
 0  1  4  9  16  25  36  49  64  81  100
Ok

300 '-- chained program --
310 PRINT "< chained program >"
320 PRINT A$
330 A%=A%+A%:B=B*B:C#=C#/C#
340 PRINT "A =";A%,"B =";B,"C =";C#
350 FOR I=0 TO 10
360  A%(I)=A%(I)*A%(I)
370  PRINT A%(I);
380 NEXT I
```

CIRCLE　N88

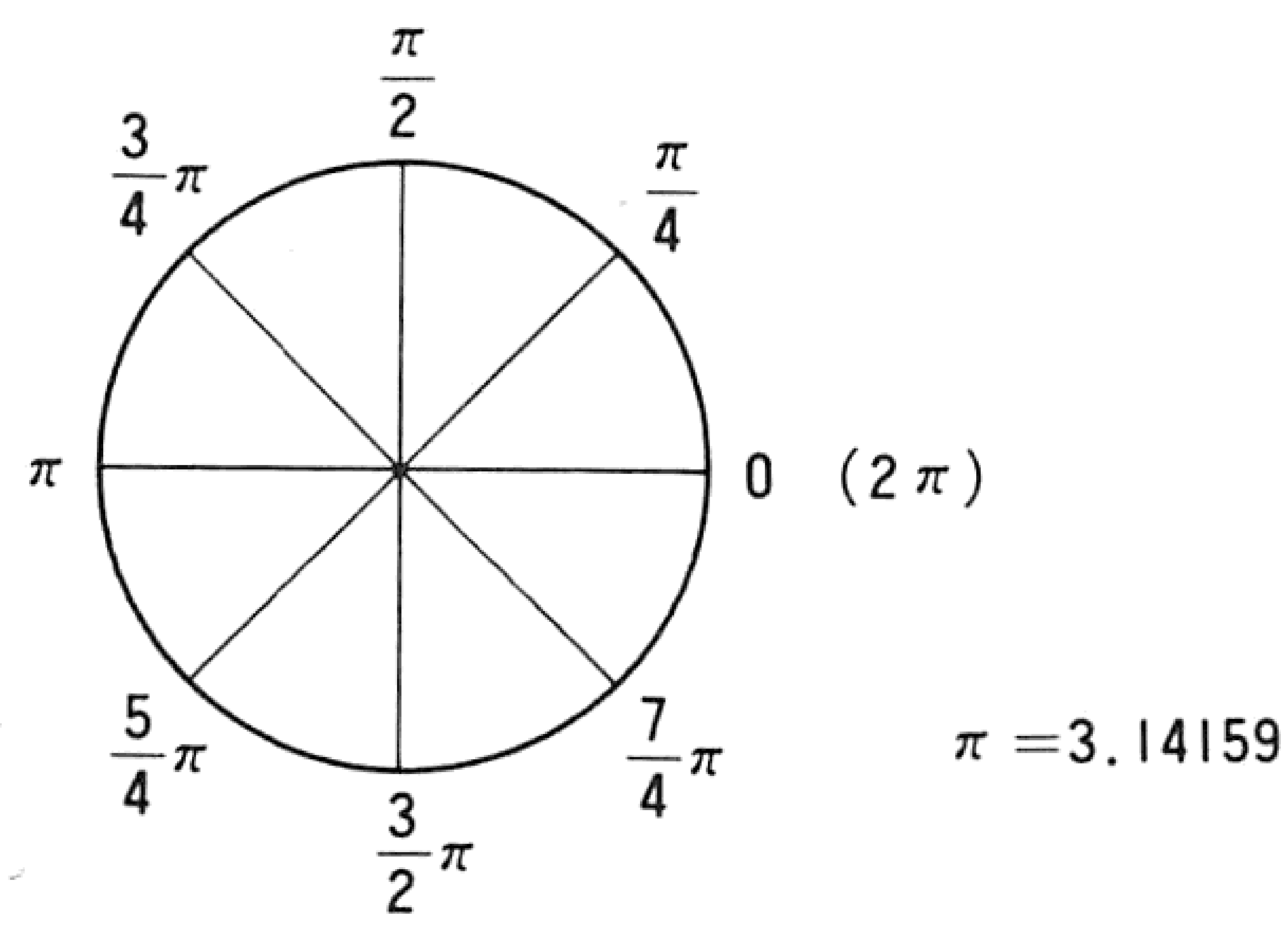
機　能	楕円（円）を描く.
書　式	CIRCLE ｛(Wx, Wy) / STEP(x, y)｝, 〈半径〉〔, 〈パレット番号〉〕〔, 〈開始角度〉〕〔, 〈終了角度〉〕〔, 〈比率〉〕
文　例	CIRCLE (80, 80), 40, 4, 0, 6.28

解　説　ワールド座標の (Wx, Wy) を中心とし，〈半径〉で指定される大きさの円を描きます．

〈パレット番号〉が指定されると，指定されたパレットの色で円を描きます．省略された場合には，COLOR 文で指定されているフォアグラウンドカラーが用いられます．

CIRCLE は通常円を描きますが，〈開始角度〉，〈終了角度〉を指定すると，指定された角度の範囲内のみに円弧を描きます．角度の指定の単位はラジアンで，省略値はそれぞれ，0 と 2π です．

〈開始角度〉，〈終了角度〉が負であった場合には，その絶対値をとり，正にした角度が用いられますが，その時中心から半径が描かれますので扇形を描くことができます．〈開始角度〉，〈終了角度〉は，-2π から $+2\pi$ の範囲内にないと〝Illegal function call″ (FC Error) のエラーとなります．



〈比率〉は，(垂直方向の半径)／(水平方向の半径) で指定します．省略された場合には，640×200ドットのモードでは0.5，640×400ドットのモードでは1.0が用いられます．

〈比率〉が 1 以下の場合には，〈半径〉は水平方向の半径を意味します．また 1 以上の場合には，〈半径〉は垂直方向の半径を意味します．つまり，〈半径〉が同じで

も〈比率〉が1以下ならば平べったい楕円となり，1以上ならば長細い楕円となります．

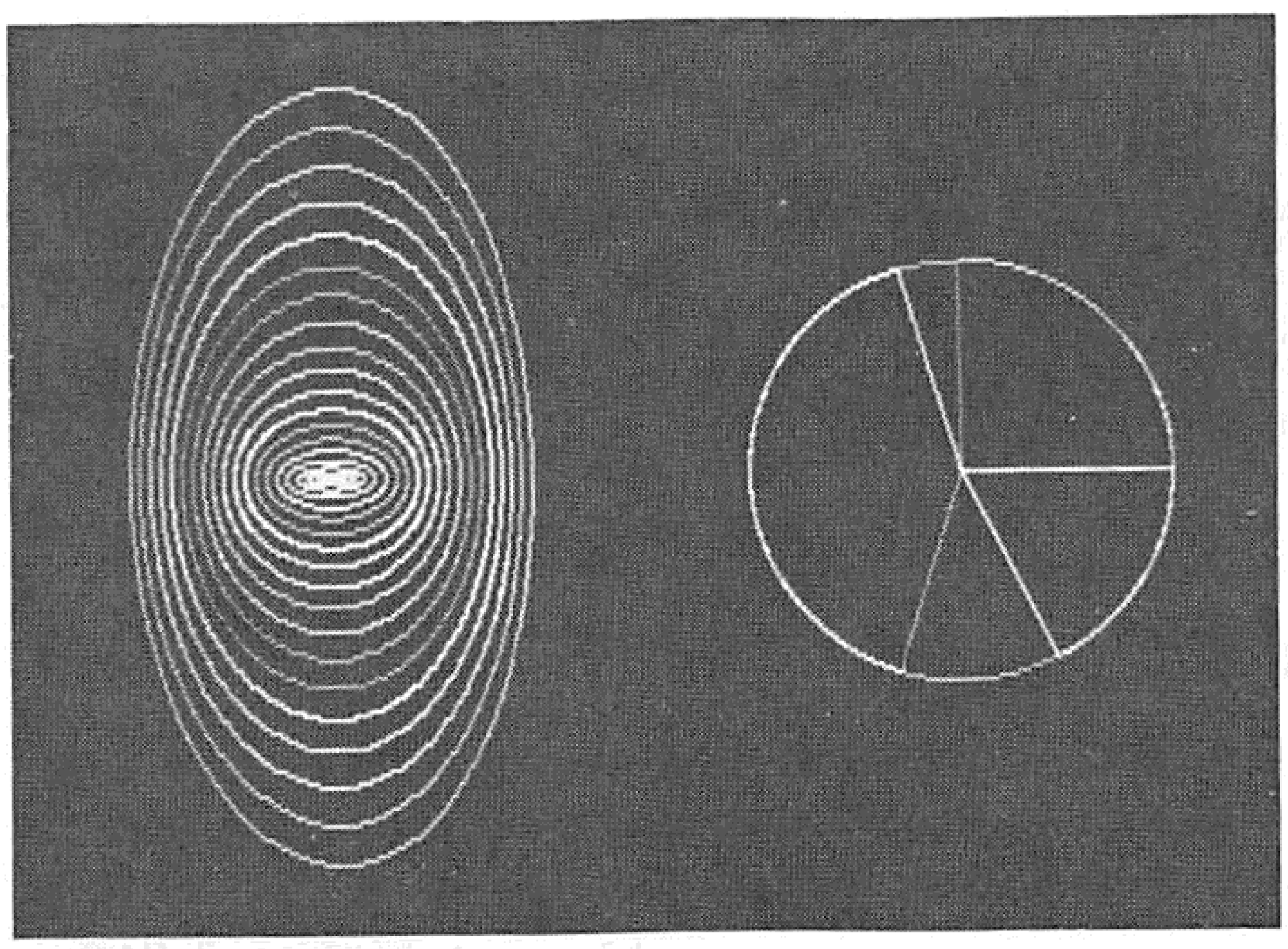
中心座標は，STEPを付けてLP（1章･座標の指定の仕方参照）からの相対座標により指定することもできます．

CIRCLEを実行すると，LPは円の中心座標（Wx，Wy）に設定されます．

参照：COLOR

サンプル プログラム

```
100 '-- circle display --
110 SCREEN 0,0:CLS 3
120 FOR R=1 TO 100 STEP 5
130  CIRCLE(150,100),R,(R MOD 7)+1,,,R/100
140 NEXT R
150 '
160 ST=.00001
170 DATA 25,5,40,13,17
180 FOR I=1 TO 5
190  READ DAT:EN=ST+DAT/100*3.14*2
200  CIRCLE(450,100),100,I,-ST,-EN
210  ST=EN
220 NEXT I
```



CLEAR　[N_{88}] [N]

機　能　変数の初期化およびメモリ領域の設定をする.

書　式　CLEAR 〔〈ストリング領域の大きさ〉〕,〔〈メモリの上限〉〕
〔〈スタックの大きさ〉〕

文　例　CLEAR , &HE500

解　説　全ての数値変数を0に，文字変数を〝〟(ヌルストリング)に初期化します.

〈メモリの上限〉が指定された場合には，指定された番地をBASICの使用するメモリの上限値としますので，その番地以降に置かれたデータや，機械語プログラムは，BASICによってクリアあるいは破壊されることはありません.

〈スタックの大きさ〉は，BASICがFOR, GOSUB, PAINT等に使用するスタック領域の大きさをバイト数で指定します.リセット時の初期化された状態では，512に設定されています.

CLEARは，DEF文(DEF FN, DEF USR, DEFINT等)によって定義あるいは指定された情報もすべて無効にします.

〈ストリング領域の大きさ〉は，N_{88}-BASICでは意味を持たないパラメータになっており，指定されても何の影響も与えません.このパラメータは，N-BASICとの構文上の統一を保つ目的で準備されています.N_{88}-BASICは，必要なストリング領域を自動的に確保します.

〈ストリング領域の大きさ〉は，N-BASICが文字列処理のために用いるメモリ領域の大きさをバイト数で指定します.多くの文字列演算を行ったり大きな文字列配列を用いる場合には，この領域を大きく指定しないと〝Out of string space〟(OS Error)のエラーになります.リセット後の初期化された状態ではこの領域の大きさは300に初期設定されています.

〈メモリの上限〉の意味はN_{88}-BASICと全く同じで，BASICの使用するメモリの上限番地を設定します.

〈スタックの大きさ〉のパラメータはN-BASICでは指定することができません.

注意：BASICで使用するスタックは，〈メモリの上限〉を起点として番地の小さい方(0に近い方)へ順に取られていきます.従ってモニタやPOKE文で〈メモリの上限〉より0に近い方の番地の内容を書きかえますと，プログラムが暴走したり，データが壊れることがあります.

サンプル
プログラム

```
100 '-- valuable clear --
110 DEF FNX(X)=X^2+X+5
120 A%=213
130 B=74.7543
140 C#=76346.43263654304#
150 D$="ABCDEFG"
160 DIM E(10)
170 FOR I=0 TO 10
180  E(I)=I
190 NEXT I
200 PRINT
210 GOSUB *DUMP
220 CLEAR
230 GOSUB *DUMP
240 END
250 '
260 *DUMP
270  PRINT "A%=";A%,"B!=";B!,"C#=";C#
280  PRINT "A$=";CHR$(&H22);D$;CHR$(&H22)
290  PRINT "E(X)=",
300  FOR I=0 TO 10
310  PRINT E(I);
320  NEXT I:PRINT
330  PRINT "FNX(X)=";FNX(10)
340  PRINT:PRINT
350 RETURN
```

```
run

A%= 213         B!= 74.7543    C#= 76346.43263654304
A$="ABCDEFG"
E(X)=            0  1  2  3  4  5  6  7  8  9  10
FNX(X)= 115


A%= 0           B!= 0          C#= 0
A$=""
E(X)=            0  0  0  0  0  0  0  0  0  0  0
FNX(X)=
Undefined user function in 330
Ok
```

CLOAD／CLOAD？ N

機　能	カセットテープからメモリへプログラムをロードする．
書　式	CLOAD <ファイル名> CLOAD？<ファイル名>
文　例	CLOAD　"DEMO" CLOAD？　"DEMO"

解　説　CLOADはカセットテープからプログラムをロードする命令です．<ファイル名>は，6文字以内で指定してください．

CLOAD？は現在メモリ上にあるプログラムと<ファイル名>で指定されたテープ上のプログラムを比較（ベリファイ）します．この時すべての内容が等しければBASICはOkとプリントし，もしそうでなければBadとプリントして実行を終了します．

このベリファイはメモリ上のプログラムがミスなくテープ上にセーブされているかを確認するためのものであり，一般にCSAVEに引き続いて行います．

参照：CSAVE

サンプルプログラム

```
cload "sample"

csave "test"
cload?"test"
```

CLOSE　N88　N　（N ではディスクのみ）

機　能	ファイルを閉じる.
書　式	CLOSE〔〔#〕<ファイル番号>〔,〔#〕<ファイル番号>〕…〕
文　例	CLOSE

解　説　<ファイル番号>に対応するファイルを閉じます. 以後, 指定された<ファイル番号>は, 異なるファイルを開くために利用できるようになります. また閉じられたファイルは, 同じあるいは異なったファイル番号によって再び開くことができます.

CLOSEでは, <ファイル番号>を複数指定することにより, 一度に複数のファイルを閉じることができます. また<ファイル番号>が省略された場合には, その時, 開いているファイルすべてを閉じます.

クローズしたファイルに対しては, 再びオープンするまで入出力を行うことはできません.

CLOSEは, ファイルが出力用にオープンされていた場合には, バッファに残っていたデータの掃き出しを行いますので, ファイルへの出力処理を正しく終了するためには, 必ずクローズを行わなくてはなりません.

END, NEWは, 自動的にすべてのファイルを閉じる処理を行います. STOP文はファイルを閉じる作業は行いませんので注意してください.

注意：ファイルがオープンされたままの状態でフロッピィディスクを取り出すと, そのフロッピィディスクの内容や次に入れたフロッピィディスクの内容を壊すことがあります。したがってフロッピィディスクを取り出す前にCLOSEを実行して下さい。

参照：OPEN, END, NEW

サンプルプログラム

```
10 OPEN "lpt1:" FOR OUTPUT AS #1
20 OPEN "cas1:" FOR OUTPUT AS #2
30 PRINT #1,"LINE PRINTER"
35 CLOSE #1
40 PRINT #2,"CASSETTE"
50 CLOSE #2
```

CLS　N88

機　能	現在アクティブな画面をクリアする.
書　式	CLS〔〈機能〉〕
文　例	CLS 2

解　説　〈機能〉は，1，2，3の値をとり，それぞれ次のように働きます．省略された場合には1が選択されます．

1）テキスト画面のみをクリアします．テキスト表示モードがCONSOLE文，COLOR文により，白黒，リバースモードになっている場合には画面は白くなります．

2）グラフィック画面のビューポート内のみをクリアします．グラフィック表示がSCREEN文によりカラーモードに設定されている場合には，COLOR文により指定されたバックグラウンドカラーで，ビューポート内をクリアします．

3）テキスト画面，グラフィック画面の両方をクリアします．

テキスト画面がクリアされる時は，CONSOLE文によって指定されたスクロールウィンドウ内のみをクリアします．またグラフィック画面をクリアした場合には，LPはビューポートの左上の頂点に移動します．

参照：SCREEN, COLOR, CONSOLE, VIEW, CMD CLS

サンプルプログラム

```
100 SCREEN 0,0
110 GOSUB *DISPLAY
120 BEEP:CLS 1
130 GOSUB *DISPLAY
140 BEEP:CLS 2
150 GOSUB *DISPLAY
160 BEEP:CLS 3
170 GOSUB *DISPLAY:VIEW(200,50)-(400,150),,7
180 BEEP:CLS 3
190 SCREEN 0:END
200 '-- display routine --
210 *DISPLAY:FOR I=1 TO 20
220  LOCATE RND*78,RND*23
230  PRINT "*"
240  LINE(RND*639,RND*199)-STEP(40,20),INT(RND*7+1),B
250 NEXT I:RETURN
```

〔1〕 COLOR　N88

機　能　ディスプレイ画面の各部の色を指定する．

書　式　COLOR 〔〈ファンクションコード〉〕〔，〈バックグラウンドカラー〉〕〔，〈ボーダーカラー〉〕〔，〈フォアグラウンドカラー〉〕

文　例　COLOR　7，0，，7

解　説　テキスト画面の文字のカラーを変えたり，グラフィック画面のフォアグラウンド，バックグラウンドのカラーを設定したりする機能があります．

〈ファンクションコード〉は，テキスト画面の文字にいろいろな機能を与えます．このファンクションコードは，現在テキスト画面がカラーモードになっているか，白黒モードになっているかによって働きが異なります．

白黒モードの場合（CONSOLE　,,,　0）

0――ノーマル（通常の表示）
1――シークレット（文字は表示されない）
2――ブリンク（点滅する）
3――シークレット（1と同じ）
4――リバース（反転する）
5――リバースシークレット（反転して文字は表示されない）
6――リバースブリンク（反転して点滅する）
7――リバースシークレット（5と同じ）

カラーモードの場合（CONSOLE　,,,　1）

0――黒	1――青	2――赤	3――紫
4――緑	5――水色	6――黄色	7――白

但し，テキスト画面の文字の色は，あくまでこのカラーコードによって表されます．後述する〝パレット番号〟ではありません．したがってパレットを変化させても文字色が変化することはありません．

これら〈ファンクションコード〉の使い方は，N-BASICのCOLOR文に用いる場合と全く同じです．

〈バックグラウンドカラー〉と〈フォアグラウンドカラー〉はグラフィック画面のカラーを設定するもので，いずれも〝パレット番号〟（1章・カラー参照）によって指定します．

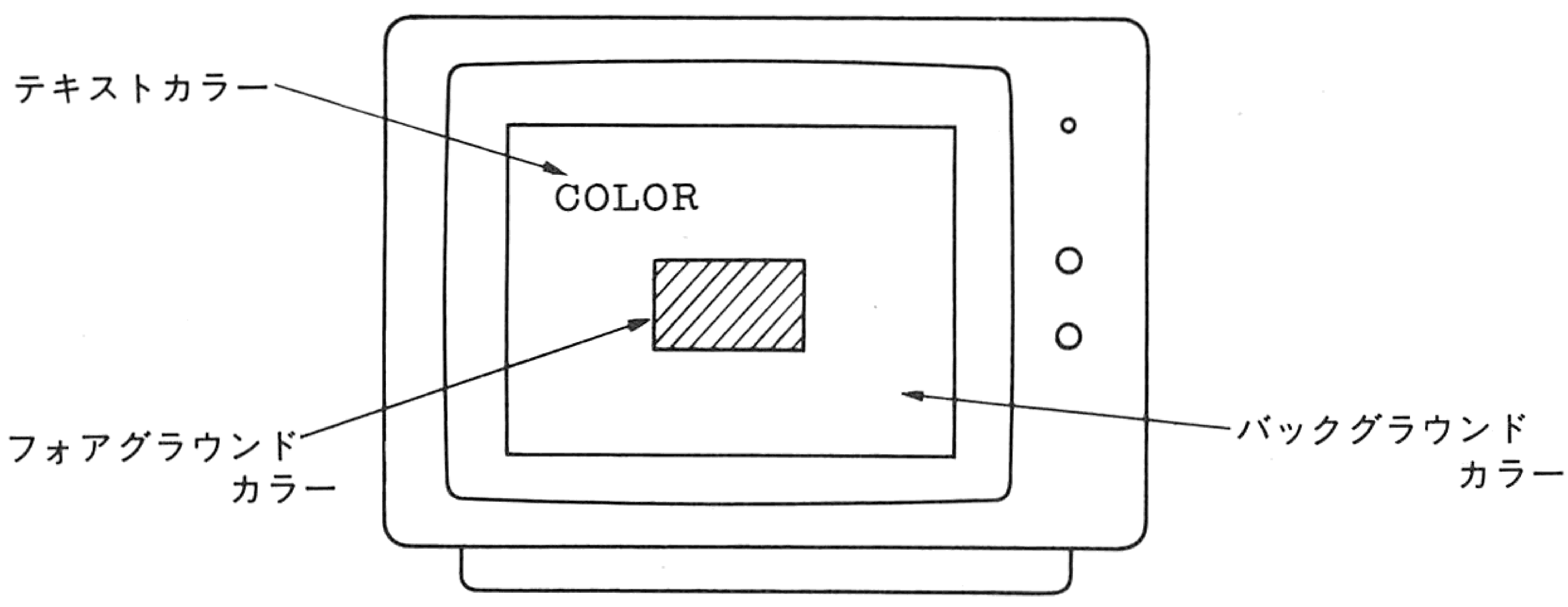
〈フォアグラウンドカラー〉とは，グラフィック画面に点や線を表示したりする

時に使われる色のことです．N_{88}-BASICの種々のグラフィック命令（PSET, LINE, CIRCLEなど）で特別に色指定をしなかった場合，このフォアグラウンドカラーが採用されます．

〈バックグラウンドカラー〉とは，グラフィック画面の地の色のことで，この命令実行後CLS命令によって画面をクリアすると，この色によって画面が塗り変えられます．また今後PRESET命令を色指定なしで実行すると，この色が採用されることになります．

〈ボーダーカラー〉は，PC-8801MKIIでは意味を持ちません。PC-8801のN_{88}-BASICとのコンパチビリティを保つためのものですので普通は省略して使ってください。

注意：このCOLOR文の4つのパラメータはすべて色を指定する時に使うものですが，前記の通りその指定の仕方が，カラーコードの場合とパレット番号の場合があり複雑です．１章の〝カラー〟を参照しカラーコードとパレット番号の関係を確実に把握しておくようにしてください．



参照：１章・カラー，〔２〕COLOR，COLOR@，CONSOLE

サンプル
プログラム

```
100 SCREEN 0,0
110 COLOR 7,4,,7:CLS 3
120 GOSUB *LINEB
130 FOR I=1 TO 7
140  COLOR I,,,I:GOSUB *LINEB
150 NEXT I:END
160 *LINEB:PRINT "COLOR"
170 LINE(100,50)-STEP(200,100),,BF
180 RETURN
```

〔2〕 COLOR　N88

機　能　カラーパレットを変更する.

書　式　COLOR＝(〈パレット番号〉, 〈カラーコード〉)

文　例　COLOR＝(2, 4)

解　説　N$_{88}$-BASICのグラフィック画面への色の指定はすべてカラーパレットによって行います. この命令は, どのパレットにどの色を対応させるかを決めるものです.

〈パレット番号〉とは, ユーザーのために用意された0から7までの8つのカラーパレットにつけられた固有の番号で, ユーザーはそれぞれのパレットにどの色を持ってくるかを〈カラーコード〉によって指定します (詳しくは1章・カラー参照). パレット番号とカラーコードの関係は次の通りです.

〈パレット番号〉		〈カラーコード〉
0		0 (黒)
1	ユーザーはどのパレットにどのカラーコードを対応させるか任意に決めることができる.	1 (青)
2		2 (赤)
3		3 (紫)
4		4 (緑)
5		5 (水色)
6		6 (黄色)
7		7 (白)

たとえば文例の命令を実行した場合, 2番のパレットは緑 (カラーコード4) が割り付けられたことになります.

BASICが起動した初期状態では, カラーパレットはそのパレット番号と同じカラーコードが割り付けられています. したがって, もしカラーパレットを全く変更しなければ, カラーパレットとカラーコードは同じものとして使ってさしつかえありません.

カラーパレット変更後は, グラフィック画面に表示されている色及びこれから画面に対して行うすべてのグラフィック命令 (PSET, LINEなど) におけるカラー表示は, そのパレット番号に割り付けられているカラーコードに設定されます.

したがってプログラム中でカラーパレットを変更した場合は, 困惑しないようにプログラム終了時にパレットを初期状態に戻しておくようにしてください.

またN$_{88}$-BASICでパレットの使用が許されているのは, グラフィック画面に

対してだけです．したがってテキスト画面のカラーはパレットの変更に影響されません．

参照：1章・カラー，〔1〕COLOR

サンプルプログラム

```
100 '-- palette alternation --
110 FOR I=1 TO 7
120  COLOR=(I,7)
130 NEXT I
140 '
150 FOR I=1 TO 7
160  LINE(0,I*20)-STEP(639,10),I,BF
170 NEXT I
180 '
190 FOR I=1 TO 7
200  FOR J=1 TO 7
210   COLOR=(J,I)
220   GOSUB *WAITSUB
230  NEXT J
240 NEXT I
250 '
260 '-- palette initialize --
270 FOR I=1 TO 7
280  COLOR=(I,I)
290 NEXT I:END
300 '-- wait routine --
310 *WAITSUB
320  FOR K=0 TO 100
330  NEXT K
340 RETURN
```

COLOR　[N]

機　能	画面に対する操作における色や機能を指定する．
書　式	COLOR 〔〈ファンクションコード〉〕〔，ヌルキャラクタコード〕 〔，〈グラフィックスイッチ〉〕
文　例	COLOR 5, 0, 1

解　説　ディスプレイ画面に対するすべての操作に使われる色や機能を設定します．

1章で解説した通り，N - BASICは画面上でテキスト画面とグラフィック画面という区別をしていません．したがって，この命令によりキャラクタ及びグラフィックのカラーが設定されます．

〈ファンクションコード〉は色や機能を設定するパラメータですが，現在CONSOLE文によって設定されているモードによって働きが異なります．これは，N$_{88}$- BASICの〔1〕COLORの〈ファンクションコード〉と全く同様ですから，そちらを参照してください．

〈ヌルキャラクタコード〉は画面をクリアした場合に表示されるキャラクタのコードを指定します．通常は0を指定します．

〈グラフィックスイッチ〉は，1にすればグラフィックモードに，0にすればキャラクタモードになります．キャラクタモードの場合でグラフィック命令の実行ができない訳ではないのですが，グラフィックモードにしておくと動作が保障されます。〈グラフィックスイッチ〉は，設定したあと，画面をクリアしないと機能しません。

参照：CONSOLE, [N88]〔1〕COLOR

サンプルプログラム

```
console ,,,0
color 4
print"N-BASIC"

console ,,,1
color 4
print"N-BASIC"
```

COLOR@ N88

機　能	**テキスト画面に書かれた文字などに色や機能を設定する.**
書　式	COLOR@ $(X_1, Y_1)-(X_2, Y_2)$〔,〈ファンクションコード〉〕
文　例	COLOR@(0,0)-(79, 4), 2

解　説　テキスト画面のキャラクタ座標の2点(X_1, Y_1), (X_2, Y_2)を対角とする四角形の領域に書かれている，文字やグラフィックキャラクタに色を付けたり，ブリンクなどの機能を設定したりします.

指定する座標(X_1, Y_1), (X_2, Y_2)は必ずキャラクタ座標でなくてはなりません. また〈ファンクションコード〉は，CONSOLE文によって設定されているモードによって持つ意味が異なります. この機能及び指定の仕方は〔1〕のCOLOR文の〈ファンクションコード〉の場合と全く同様ですから，そちらを参照してください.

もし〈ファンクションコード〉が省略された場合は，7を値として採用します.

注意：この機能はあくまでテキスト画面に書かれている文字などに対して有効です. したがって何も書かれていない場合は，何の作用もありません. またこの命令によって設定された領域の上に新しく文字などを書いた場合，書かれた文字はこの命令の影響を受けません.

参照：CONSOLE,〔1〕COLOR

サンプルプログラム

```
100 CONSOLE 0,25,0,1
110 WIDTH 80,25:CLS
120 FOR I=0 TO 31
130   PRINT "PC-8801MK2";
140 NEXT I
150 FOR I=0 TO 7
160   COLOR@(I*10,0)-(I*10+9,4),I
170 NEXT I
```

COMMON（ディスク） N88

機　能	CHAIN 文が実行された時変数を引き渡す．
書　式	COMMON <変数名>〔，<変数名> …〕
文　例	COMMON A, B, XY(), NA$

解　説　CHAIN 文が実行され，メモリ上のプログラムと外部からのプログラムがチェインされた時，メモリ上のプログラムの変数を引き渡します．したがってCOMMON 文は必ずCHAIN 文とペアになって使われます．

COMMON 文は非実行文ですから，プログラム中のどこにあってもかまいませんが，前にある方が望ましく，少なくとも対応するCHAIN 文の前にある必要があります．

<変数名>は1行の許す範囲で何個でも並べることができますが，1つのプログラム中のCOMMON 文に同じ変数名があってはいけません．また配列変数名は〝()〟を付け加えて表現します．

もしすべての変数を引き渡したいのであれば，CHAIN 文のALL オプションを使う方が便利です．

参照：CHAIN

サンプルプログラム

```
100 DIM E(10)
110 COMMON A,B%,C#,D$,E()
120 A=546
130 B%=123
140 C#=1.23456789#
150 D$="abcde"
160 E(0)=10
170 CHAIN "1:test"
180 END
```

COM ON／OFF／STOP　N88

機　能　通信回線からの割り込みの許可，禁止，停止を設定する．

書　式　1）COM ON

2）COM OFF

3）COM STOP

文　例　COM ON

解　説　コミュニケーションポート(RS - 232C 回線)に外部からの通信が入ったことによる割り込みを，許可(ON)するか，禁止(OFF)するか，停止(STOP)するかを設定します．

1）割り込みを許可します．この命令実行後は，コミュニケーションポートに通信が入るごとに割り込みを発生し，ON COM GOSUB 文で定義されている処理ルーチンに分岐します．

2）割り込みを禁止します．この命令実行後は，通信があっても処理ルーチンへの分岐は起こりません．

3）割り込みを停止します．この命令実行後は，通信があってもそのことを覚えているだけで処理ルーチンへの分岐は起こりません．しかし，その後 COM ON によって割り込みが許可されると，先程の通信によって処理ルーチンに分岐します．

注意：プログラム終了時には COM は OFF の状態にしておいてください．

参照：APPENDIX, ON COM GOSUB, KEY ON/OFF/STOP

サンプルプログラム

```
COM ON
COM OFF
COM STOP
```

CONSOLE　N88　N

機　能　テキスト画面のモード設定を行う．

書　式　CONSOLE〔〈スクロール開始行〉〕，〔〈スクロール行数〉〕〔，〈ファンクションキー表示スイッチ〉〕〔，〈カラー／白黒スイッチ〉〕

文　例　CONSOLE 0，25，0，1

解　説　〈スクロール開始行〉と〈スクロール行数〉を指定することによって，画面上でスクロールする領域（スクロールウィンドウ）を指定します．

CLS 及び PRINT CHR$(12)は，このスクロールウィンドウに対して実行されます．

但し，N-BASIC では，PRINT CHR$(12)は，画面全体に対して実行されます．

〈ファンクションキー表示スイッチ〉に 1 を指定すると，画面最下行にその時点でプログラマブル・ファンクションキーに登録されている文字列を表示します．0 を指定した場合，この表示を禁止します．

〈カラー／白黒スイッチ〉を 1 に指定すると，テキスト画面をカラーモードにし，0 を指定した場合には白黒モードにします．

注意：CONSOLE文を使って，テキスト画面のカラー/モノクロモードを切り替えるとSCREEN文のパラメータ4がリセットされます．(SCREEN文参照)

サンプルプログラム

```
100 '-- scroll window --
110 CLS
120 CONSOLE 0,25,0,1
130 FOR I=1 TO 25 STEP 4
140  CONSOLE 0,I
150  GOSUB *PRINTSUB
160 NEXT I
170 END
180 *PRINTSUB
190  CLS
200  FOR J=1 TO 40
210   PRINT J
220   FOR K=0 TO 10:NEXT K
230  NEXT J
240 RETURN
```

CONT　N88　N

機　能	ストップキー入力，またはSTOP文によって停止したプログラムの実行を再開する.
書　式	CONT
文　例	CONT

解　説　CONTは通常デバッグのためにSTOP文と共に用いられます. STOP文やストップキーの入力によりプログラムの実行を停止し，ダイレクトモードで変数の値等を調べたり変更した後に，CONT文によって実行を再開します.

但し，実行停止中にプログラム内容の変更を行った場合には，CONT文による継続実行はできません.

サンプルプログラム

```
100 FOR I=1 TO 9
110   PRINT I,SQR(I)
120 NEXT I
130 END

run
 1             1
 2             1.41421
 3             1.73205
 4             2
 5             2.23607
^C
Break in 110
Ok
print i
 5
Ok
cont
 6             2.44949
 7             2.64575
 8             2.82843
 9             3
Ok
```

COPY　N88

機　能	画面上に表示されている文字，図形をプリンタに出力する．
書　式	COPY〔<機能>〕
文　例	COPY 3

解　説　<機能>は，1から5の値をとり，それぞれ次のように働きます．省略された場合には1が選択されます．

1：テキスト画面のみをプリンタにコピーします．
2：グラフィック画面のみをプリンタにコピーします．
3：テキスト画面，グラフィック画面の両方をプリンタにコピーします．
4：グラフィック画面のみをプリンタにコピーします(漢字出力用)．640×200ドットのモードで出力された漢字（グラフィックパターン）はこのモードで出力すると上下が2分の1に縮小されて，美しくコピーされます．
5：テキスト画面，グラフィック画面の両方をプリンタにコピーします．4と同じく640×200のモードで出力されたパターンは2分の1に縮小されます．

注意：グラフィック画面のコピーは，専用プリンタ以外のプリンタではできないことがありますので注意してください。

COPY文(COPYキー)でグラフィック画面をプリンタへコピーすると，画面とプリンタとの縦横の比が異なる場合があります．

サンプルプログラム

```
100 SCREEN 0,0:CLS 3
110 VIEW(1,1)-(638,198),,7
120 GOSUB *DISPLAY
130 FOR I=1 TO 5
140  COPY I
150 NEXT I
160 END
170 *DISPLAY
180  FOR J=1 TO 30
190  LOCATE RND*70,RND*22+1:PRINT "PC-8801MK2"
200  LINE(RND*639,RND*199)-STEP(40,20),INT(RND*7+1),BF
210  CIRCLE(RND*639,RND*199),40,INT(RND*7+1)
220 NEXT J:RETURN
```

PC-8801MK2

PC-8801MK2 PC-8801MK2

PC-8801MK2

PC-8801MK2

PC-8801MK2 PC-8801MK2

PC-8801MK2 PC-8801MK2

PC-8801MK2 PC-8801MK2

PC-8801MK2 PC-8801MK2

PC-8801MK2 PC-8801MK2 PC-8801MK2

PC-8801MK2 PC-8801MK2

PC-8801MK2

PC-8801MK2

PC-8801MK2

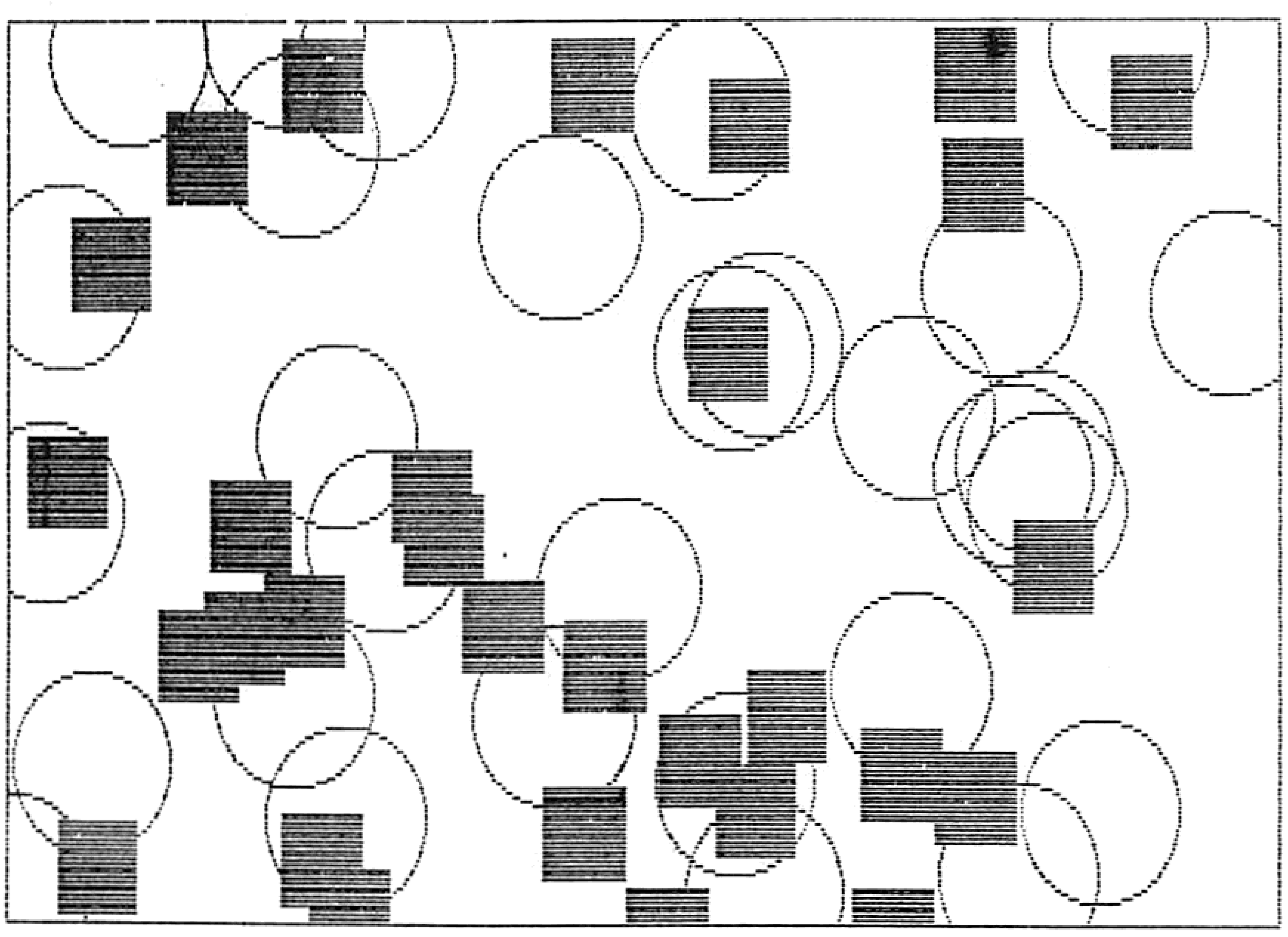
PC-8801MK2

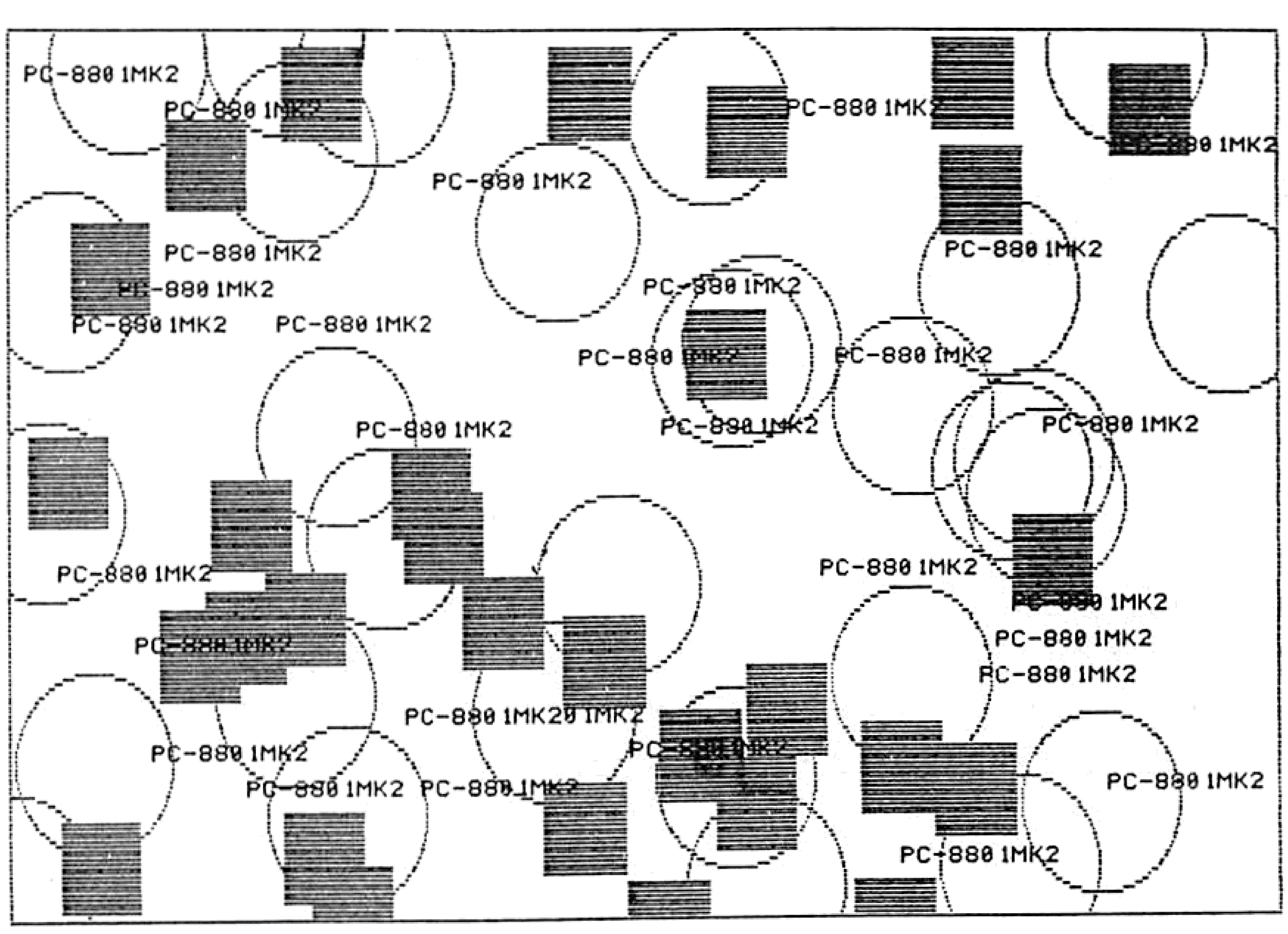
PC-8801MK20 1MK2

PC-8801MK2 PC-8801MK2

PC-8801MK2 PC-8801MK2 PC-8801MK2

PC-8801MK2





CSAVE　[N]

機　能	**メモリ上にあるプログラムをカセットテープへセーブする.**
書　式	CSAVE　<ファイル名>
文　例	CSAVE　"DEMO1"

解　説　メモリ上にあるプログラムをカセットテープへ記録（セーブ）します.

テープ上にセーブされたプログラムは，ファイル名により区別されます．この<ファイル名>は最大6文字からなる文字列によって指定します.

参照：CLOAD

サンプルプログラム

```
csave"GAME1"
cload?"GAME1"
```

DATA　N88　N

機　能	READ 文で読み込まれる数値，文字定数を格納する.
書　式	DATA 〈定数〉〔，〈定数〉…〕
文　例	DATA　1，NEC，8801MK2

解　説　DATA 文は非実行文で，プログラム中のどこに置いてもかまいません．1つの DATA 文には，1行（255文字）に入るだけのデータをセットすることができます．また，1つのプログラム内には，任意の数のデータ文を置くことができます.

READ 文は行番号の小さい方から順番に，DATA 文中のデータを読み込んでいきます.

〈定数〉は，任意の型の定数，即ち，整数，単精度実数，倍精度実数，文字定数のいずれでもかまいません．但し，定数式（2 * 3など）は許されません．また READ 文で読み込まれる場合，READ 文で指定されている変数の型は，対応する DATA 文の定数の型と一致しなければなりません.

DATA 文中のデータはコンマ(,)によって区切られますが，文字定数で，その文字列の先頭，または最後がコンマやピリオド(.)，意味のある空白である場合には，その文字定数の前後をダブルクォーツ（"）でくくる必要があります.

RESTORE 文によって READ 文で読み込む DATA 文を行単位で指定する事ができます.

参照：READ，RESTORE

サンプルプログラム

```
100 DATA 10,10,   NEC-8801MK2,"   NEC-8801MK2"
110 READ A
120 READ B$:B=VAL("&h"+B$)
130 READ C$:READ D$
140 PRINT A:PRINT B
150 PRINT C$:PRINT D$
```

```
run
 10
 16
NEC-8801MK2
   NEC-8801MK2
Ok
```

DEF FN　N88　N

機　　能	ユーザーによって作られた関数を定義する.
書　　式	DEF FN〈名前〉〔(〈パラメータリスト〉)〕=〈関数の定義式〉
文　　例	DEF FNA(X, Y)=X ＊ 2+Y ＊ 3

解　　説　関数の名前は〝FN〟とこれに続く〈名前〉を含めたものであり,〈名前〉は変数名として正しい形をしたものでなければなりません(N_{88}- BASIC では40文字まで, N - BASIC では2文字までです).

〈パラメータリスト〉はその関数が呼ばれた時に,〈関数の定義式〉中の同じ名前の変数に対応し, この変数名は,〈関数の定義式〉を評価する際にのみ有効となり, プログラム中に同一名の変数があっても影響はありません. また,〈関数の定義式〉中で使われている変数が,〈パラメータリスト〉中にない場合は, その変数が, その時点で持っている値が使われます.

〈関数の定義式〉は, その関数の演算内容を記述する式で, 1行の範囲に限られます. このユーザー定義関数は, 数値関数, 文字関数, その混在のいずれでもかまいませんが,〈パラメータリスト〉内の変数と, それに渡される変数とが同一の型でなければなりません.

DEF FN 文は非実行文ですが, これによって定義される関数がプログラム中で呼ばれる前に実行されていなければなりません. また, ダイレクトモードでの実行はできません.

参照：変数名と型宣言

サンプルプログラム

```
100 DEF FNFUKURI(G,R,N)=G*(1+R)^N
110 DEF FNTANRI(G,R,N)=G*(1+R*N)
120 INPUT "ガンキン   :",GANKIN
130 INPUT "ネンリ     :",NENRI:INPUT "ネンスウ    :";NEN
140 FGOKEI=FNFUKURI(GANKIN,NENRI,NEN)
150 TGOKEI=FNTANRI(GANKIN,NENRI,NEN)
160 F$="¥¥###########,._-"
170 PRINT "フクリ デハ ";USING F$;FGOKEI
180 PRINT "タンリ デハ ";USING F$;TGOKEI

run
ガンキン   :100000
ネンリ     :.07
ネンスウ   :? 10
フクリ デハ        ¥196,715.-
タンリ デハ        ¥170,000.-
Ok
```

DEFINT/SNG/DBL/STR　N88 N

機　能	変数の型宣言をする.
書　式	DEF INT / SNG / DBL / STR　<文字の範囲>
文　例	DEFINT A, I－K

解　説　<文字の範囲>で指定された文字で始まる変数名の変数を，DEFINTでは整数型に，DEFSNGでは単精度実数型，DEFDBLでは倍精度実数型に，DEFSTRでは文字型にそれぞれ定義します．

<文字の範囲>で指定される文字は英字1文字で，複数個指定する場合はハイフンでつないでその範囲を示します．

但し，これらによって行われた型宣言より，型宣言文字による指定の方が優先されます．また，型宣言が行なわれていない文字で始まる変数は，すべて単精度変数とみなされます．

サンプルプログラム

```
100 DEFINT I-M
110 DEFSNG A,B
120 DEFDBL D
130 I1=1.23:J=65.643
140 ABC=1.23:BBB=65.643
150 D=3.141592654000001#
160 D%=3.141592654000001#
170 PRINT "I1 =";I1,,"J =";J
180 PRINT "ABC =";ABC,,"BBB =";BBB
190 PRINT "D =";D,"D% =";D%
200 END

run
I1 = 1                         J = 66
ABC = 1.23                     BBB = 65.643
D = 3.141592654000001          D% = 3
Ok
```

DEF USR　N88　N

機　能	**機械語で作られたユーザー関数の実行開始番地を定義する.**
書　式	DEF USR〔〈番号〉〕=〈開始番地〉
文　例	DEF USR3=&HE000

解　説　USR 関数（ユーザー関数）が呼び出す，機械語ルーチンの実行開始番地の設定を行います．〈番号〉は，0～9までの値で，複数のユーザー関数を用いる場合の識別を行います．したがって最大10個までのユーザー関数を準備，利用することができます．〈番号〉が省略された場合には0と解釈されます．

〈開始番地〉は〈番号〉によって指定されるユーザー関数の実行開始番地を指定します．

USR 関数の作り方については，APPENDIX・機械語プログラムの作り方を参照してください．

参照：USR，APPENDIX・機械語プログラムの作り方

サンプルプログラム

```
100 CLEAR ,&HE000
110 DEF USR=&HE000
120 DEF USR1=&HE200
130 PRINT USR(100)
140 A=USR1(1.23)
```

DELETE　N88 N

機　能	プログラムの部分削除.
書　式	DELETE〔<始点行番号>〕〔—<終点行番号>〕
文　例	DELETE 10 - 230

解　説　<始点行番号> から <終点行番号> までのプログラムを削除します.
<始点行番号> だけを指定した場合はその行だけを削除し, ハイフン以下 <終点行番号> を指定した場合には, プログラムの先頭から指定行までを削除します.

サンプルプログラム

```
100 ' DELETE ｺﾏﾝﾄﾞ ﾉ ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ
110 '
120 ' DELETE ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｦ
130 ' ｻｸｼﾞｮ ｼﾏｽ.
140 '
150 PRINT " DELETE ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｦ"
160 PRINT " ｻｸｼﾞｮ ｼﾏｽ."
170 END
Ok
delete 110-130
Ok
list
100 ' DELETE ｺﾏﾝﾄﾞ ﾉ ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ
140 '
150 PRINT " DELETE ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｦ"
160 PRINT " ｻｸｼﾞｮ ｼﾏｽ."
170 END
Ok
run
 DELETE ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｦ
 ｻｸｼﾞｮ ｼﾏｽ.
Ok
```

DIM　N88　N

機　能	配列変数の要素の大きさを指定し，メモリ領域に割り当てる．
書　式	DIM <変数名>（<添字の最大値>〔，<添字の最大値> …〕）
文　例	DIM A（12，2）

解　説　配列変数の添字の最大値を設定し，同時にメモリ上にその配列の領域を割り当てます．DIM 文で宣言せずに配列変数を用いた場合，その添字の最大値は10とみなされます．もし，設定された最大値より大きい値の添字が用いられた場合は，〝Subscript out of range〟（BS Error）エラーが起こります．

添字の最小値は，OPTION BASE 文によって，0か1に指定することができます．

但し，N－BASIC では添字の最小値は常に0です．

また，DIM 文が実行された時点で，その配列のすべての要素の値は0（文字型の場合はヌルストリング）に設定されます．

参照：OPTION BASE

サンプルプログラム

```
100 ' ｸｸ ﾉ ﾋｮｳ ｦ ﾂｸﾙ
110 OPTION BASE 1
120 DIM KUKU(9,9)
130 FOR I=1 TO 9:FOR J=1 TO 9
140   KUKU(I,J)=I*J
150 NEXT J,I
160 FOR I=1 TO 9:FOR J=1 TO 9
170   PRINT USING "####";KUKU(J,I);
180  NEXT J:PRINT
190 NEXT I
200 END

run
   1   2   3   4   5   6   7   8   9
   2   4   6   8  10  12  14  16  18
   3   6   9  12  15  18  21  24  27
   4   8  12  16  20  24  28  32  36
   5  10  15  20  25  30  35  40  45
   6  12  18  24  30  36  42  48  54
   7  14  21  28  35  42  49  56  63
   8  16  24  32  40  48  56  64  72
   9  18  27  36  45  54  63  72  81
Ok
```

DSKO$（ディスク）　N88　N

機　能	ディスクに対しての直接書き込み．
書　式	1）DSKO$〈ドライブ番号〉，〈トラック番号〉，〈セクタ番号〉
	2）DSKO$〈ドライブ番号〉，〈サーフェス番号〉，〈トラック番号〉，〈セクタ番号〉
文　例	1）DSKO$　2，10，1
	2）DSKO$　1，0，19，1

解　説　通常のファイル操作とは無関係に，ディスク上の指定したセクタに直接書き込みを行います．この文は，既存のディスクファイルを壊す恐れがありますので，ディスク及びファイルの構成を正しく理解した上で用いてください．これらについては，ユーザース・マニュアルで説明されています．

DSKO$は，そのパラメータにより指定されたセクタに，0番バッファの内容256バイトを書き込みます．0番バッファは，通常のディスクファイル操作の際には用いることはできませんが，DSKI$とDSKO$による直接操作の際にのみ，ユーザーが用いることができます．

書き込むデータの準備は，ランダムアクセスの場合と同じようにFIELD文により，0番バッファへ変数領域を割り付けた後，LSET，RSETにより行います．あるいは，VARPTR関数により，0番バッファの置かれているメモリ番地を調べた上で，POKE文によりバッファ中にデータを準備することもできます．

〈トラック番号〉，〈セクタ番号〉として指定できる値の範囲は，指定された〈ドライブ番号〉のディスクドライブの種類によって異なりますが，BASICは，これらの値の範囲を自動的に調べ，不当であった場合には〝Illegal function call〟エラーとします．詳しくはユーザース・マニュアルを参照してください．

〈サーフェス番号〉は，指定された〈ドライブ番号〉のディスクが両面ディスクの場合にのみ必要で，0または1で指定します．

DSKO$は対象とするディスクドライブが片面ディスクであるか両面ディスクであるかによってダイナミックに書式が変ります．

1）の書式は片面ディスクを対象とする場合であり，〈サーフェス番号〉の指定は必要ありません．

2）の書式は両面ディスクを対象とする場合であり，〈サーフェス番号〉を指定しなければなりません．

注意：複数のディスクが接続されている場合，それぞれのディスクの諸元については DSKF により調べることができます．詳しくはユーザース・マニュアルを参照ください．

参照：DSKI$，FIELD，DSKF，VARPTR

サンプルプログラム

```
100 '-----        Write to the disk          -----
110 '*****  コノ program ハ disk ニ カキコミマスカラ  *****
120 '*****        フヨウイニ シヨウシナイテ゛ クタ゛サイ。      *****
130  FIELD #0,128 AS A$,128 AS B$
140  LSET A$=STRING$(128,CHR$(&HFF))
150  LSET B$=STRING$(128,CHR$(&HFF))
160 *START
170  INPUT "Drive";DR
180  IF DR<1 OR DR>8 THEN *START
190  IF DSKF(DR,2)=0 THEN 230
200  INPUT "Surface";SU
210  IF SU<0 OR SU>1 THEN 200
220 *REENT
230  INPUT "Track,Sector";TR,SE
240  IF TR<0 OR TR>DSKF(DR,0) THEN *REENT
250  IF SE<1 OR SE>DSKF(DR,1) THEN *REENT
260  IF DSKF(DR,2)=0 THEN *SINGLE
270  DSKO$ DR,SU,TR,SE:GOTO *EXIT
280 *SINGLE
290  DSKO$ DR,TR,SE
300 *EXIT:END
```

EDIT　N88

機　能	指定された行を画面上へ表示し，カーソルを移動する．
書　式	EDIT | 〈行番号〉 | 　　　| 　．　　|
文　例	EDIT 30

解　説　画面上での編集ができるように，指定された行を表示し，カーソルをその行の先頭に置きます．エラー発生時や，STOP を実行した直後に〝EDIT　.〟を行うと，エラーの発生した行，あるいは実行した STOP 文を含む行が表示され，ただちに修正作業に入ることができます．

EDIT コマンド実行中は，ロールアップ，ロールダウンキーにより指定された行の前後の表示，編集することもできます．(ディスクバージョン)

Pオプション付きでセーブされたファイルをロードして EDIT コマンドにより，内容を表示，変更しようとした場合には，〝Illegal function call〟(FC Error) のエラーとなります．

E

参照：SAVE

サンプルプログラム

```
EDIT 100
EDIT *START
EDIT .
```

END　N88　N

機能　プログラムの終了を宣言する.

書式　END

文例　END

解説　プログラムの実行を終了し，すべてのファイルをクローズしてコマンドレベルに戻ります.

END 文はプログラムの実行を終了させたい所に置けばよく，また，いくつ置いてもかまいません.プログラムの最後の END 文は省略することができますが，この場合，ファイルのクローズは行われません.

サンプルプログラム

```
100  WIDTH 80,25
110   COLOR 7
120   CLS
130 *LOOP
140   PRINT TIME$
150   A$=INKEY$
160    IF A$="s" THEN END
170 GOTO *LOOP

run
08:15:46
08:15:46
08:15:46
08:15:46
08:15:46
08:15:46
08:15:46
08:15:46
08:15:46
08:15:46
08:15:47
08:15:47
08:15:47
Ok
```

ERASE　N88 N

機　能	配列変数を消去する.
書　式	ERASE <配列変数名>〔, <配列変数名> …〕
文　例	ERASE C, D

解　説　指定した配列変数を消去します. 以後, 同一変数名の配列変数をDIM文で宣言することができます. もしERASE文を実行せずに同一変数名で宣言すると, "Redimensioned array" (DD Error : Duplicate Definition) エラーが起こります.

また, その配列変数に割り当てられていたメモリエリアを, 他の目的に使用することもできます.

参照 : DIM

サンプルプログラム

```
100 DIM A(100),B(100)
110 ERASE A
120 DIM A(200)
130 DIM B(200) :' センゲン デキナイ!!

run
Duplicate Definition in 130
Ok
```

ERROR　N88　N

機　能	**エラー発生のシミュレート，エラー番号のユーザー定義をする．**
書　式	ERROR 〈整数表記〉
文　例	ERROR 200

解　説　〈整数表記〉の値は0から255までの範囲の数を指定します．この値が，すでにBASICでエラーコードとして使われている場合，ERROR文は，そのエラーの発生をシミュレートし,対応するエラーメッセージがプリントされます．

また，未定義のエラーコードのエラーをプログラム中で条件に応じてERROR文で発生させ，ON ERROR GOTO文によって指定されたエラー処理ルーチン内で，ユーザーが定義したエラーコードとして使用することができます．

参照：ON ERROR GOTO，ERL/ERR，エラーコード表

サンプルプログラム

```
100  ON ERROR GOTO *ERMESSAGE
110 *NYURYOKU
120  INPUT "3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｾﾖ:",N
130   IF FIX(N)<>N THEN ERROR 251
140   IF N<100 OR N>999 THEN ERROR 250
150  PRINT USING "<###>";N
160 END
170 *ERMESSAGE
180  IF ERR=250 THEN PRINT " 3 ｹﾀ ﾄﾊ 100 ｶﾗ 999 !!!"
190  IF ERR=251 THEN PRINT " ｼﾞｯｽｳ ﾊ ﾆｭｳﾘﾖｸ ﾃﾞｷﾏｾﾝ "
200 PRINT
210 RESUME *NYURYOKU

run
3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｾﾖ:12876
 3 ｹﾀ ﾄﾊ 100 ｶﾗ 999 !!!

3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｾﾖ:12.4
 ｼﾞｯｽｳ ﾊ ﾆｭｳﾘﾖｸ ﾃﾞｷﾏｾﾝ

3 ｹﾀ ﾉ ｾｲｽｳ(+) ｦ ﾆｭｳﾘｮｸ ｾﾖ:145
<145>
Ok
```

FIELD　N88　N　(Nではディスクのみ)

機能　ランダムファイル・バッファに変数領域を割り当てる.

書式　FIELD〔#〕<ファイル番号>, <フィールド幅> AS <文字変数>
〔, <フィールド幅> AS <文字変数> …〕

文例　FIELD #1, 128 AS A$, 64 AS B$, 64 AS C$

解説　ファイルバッファをランダムファイル・バッファとして使用するために，ファイルバッファの中に文字変数の領域を割り当てます．

PUT 文，GET 文を使ってランダムファイルのアクセスを行う前に必ず，FIELD 文が実行されていなければなりません．

<ファイル番号>は，OPEN 文によってファイルをオープンしたときに指定した番号です．

<フィールド幅>は<文字変数>に割り当てる文字数です．1つのバッファには文字数の合計が256文字までならば，いくつの文字変数を割り当ててもかまいません．また，1つのバッファに対して，何回かに分けて FIELD 文を実行してもかまいません．

FIELD 文で指定した変数は，FIELD 文を実行した時点で，指定した文字数のヌルキャラクタ（アスキーコードが0のキャラクタ）がセットされます．また，この変数を INPUT 文，LET 文の左辺で使用しないでください．もし使用すると，その変数は，一般の文字変数領域に登録され，FIELD 文でのバッファの割り当てが無効となり，正しいファイルの入出力が行われなくなります．

F

注意：FIELD 文が1行(256文字)に書き切れない場合はダミーの変数を使って複数の行で定義します。
〔サンプルプログラム〕の行番号110～130の書き方を参照してください。

参照：OPEN，GET，PUT，LSET/RSET

サンプル
プログラム

```
100 OPEN "rtest.dat" AS #1
110 FIELD #1,30 AS A$,30 AS B$,30 AS C$
120 '-- DUMMY$ ﾊ A$+B$+C$ ﾉﾀﾒﾉ ﾀﾞﾐｰ ﾃﾞｽ｡ --
130 FIELD #1,90 AS DUMMY$,2 AS D$,4 AS E$,8 AS F$
140 LSET A$="NEC-8801MK2"
150 LSET B$="Personal"
160 LSET C$="Computer"
170 RSET D$=MKI$(8801)
180 RSET E$=MKS$(228000!)
190 RSET F$=MKD$(6.626180000000021D-34)
200 PUT #1,1
210 GET #1,1
220 PRINT A$
230 PRINT B$
240 PRINT C$
250 D=CVI(D$)
260 E=CVS(E$)
270 F=CVD(F$)
280 PRINT D:PRINT E:PRINT F
290 END
```

```
run
NEC-8801MK2
Personal
Computer
 8801
 228000
 6.62618E-34
OK
```

FILES/LFILES（ディスク）　N88　N

機　能	ディスクに入っているファイルの名前，種類と大きさを表示する．
書　式	1) FILES 〔〈ドライブ番号〉〕
	2) LFILES 〔〈ドライブ番号〉〕
文　例	FILES 2

解　説　〈ドライブ番号〉で指定された，ディスク上に登録されているファイルの名前，種類と，その大きさを表示します．1)の場合ディスプレイに，2)の場合プリンタに出力します．ファイルの大きさは，クラスタという単位で表示されます．クラスタについては，ユーザース・マニュアルを参照してください．

〈ドライブ番号〉が省略された場合には，ドライブ1が選択されます．

ファイルの種類は，出力されるファイルの名前と，タイプとの区切り文字によって表されます．区切り記号が空白の場合には，そのファイルは，アスキーセーブされたプログラムファイル，あるいは，OPEN 文によって作られた，BASIC のデータファイルであることを示します．区切り文字が，ピリオドの場合には，バイナリセーブによって作られたプログラムファイル，アスタリスクの場合には，BSAVE によって作られた機械語ファイルであることを表します．

N - BASIC には BSAVE はありませんから，区切り記号としてアスタリスクが表示されることはありません．

参照：SAVE，BSAVE

サンプルプログラム

```
files
rword        1    adump      1
hexdum p     1    errmes     1
search 1     1    pltdem o   1
viewde mo    1    tilete st  1
dskdum p     1
Ok
```

FOR…TO…STEP～NEXT　[N88] [N]

機　能　FOR文からNEXT文までの一連の命令を指定回数だけ繰り返して実行する.

書　式

```
FOR 〈変数名〉=〈初期値〉 TO 〈終値〉〔STEP 〈増分〉〕
    ≀
NEXT 〔〔〈変数名〉〕〔，〈変数名〉〕〕
```

文　例

```
FOR J=0 TO 100 STEP 2
    ≀
NEXT J
```

解　説　〈変数〉はカウンタとして使われ，最初に初期値に設定されます．そして，FOR文以降からNEXT文までが実行され，カウンタの値はSTEPによって指定された〈増分〉だけ増減されます．次にカウンタの値が〈終値〉と比べられ，カウンタの値が〈終値〉に達していなければ，FOR文の次の文へ戻り，同じ処理が繰り返されます．

STEP文が省略された場合，〈増分〉は＋1とみなされます．また，〈増分〉は，負の値をとることもできます．

次の場合には，FOR～NEXT文は実行されずに，NEXT文の次の文へ実行が移ります．

1）〈増分〉が正の値で，〈初期値〉が〈終値〉より大きい場合．

2）〈増分〉が負の値で，〈初期値〉が〈終値〉より小さい場合．

3）〈初期値〉と〈終値〉が同じ値で〈増分〉が0の場合．

但し，〈変数〉には〈初期値〉が代入されています．

また，N-BASICでは，上記の3つの場合，〈変数〉に〈初期値〉が代入され，1回だけNEXT文までの一連の命令が実行されます．

FOR～NEXTは入れ子構造にすることができます．これは1つのFOR～NEXTのなかにもう一つのFOR～NEXTを置くことができるということで，この場合，それぞれの〈変数〉は別のものを使わなければなりません．また，1つのFOR～NEXTは完全に他のFOR～NEXTの内部になければなりません．

FOR文とNEXT文とは必ず1対1に対応していなければなりません．したがって，IF文中にNEXT文を置く事ができない場合があります．

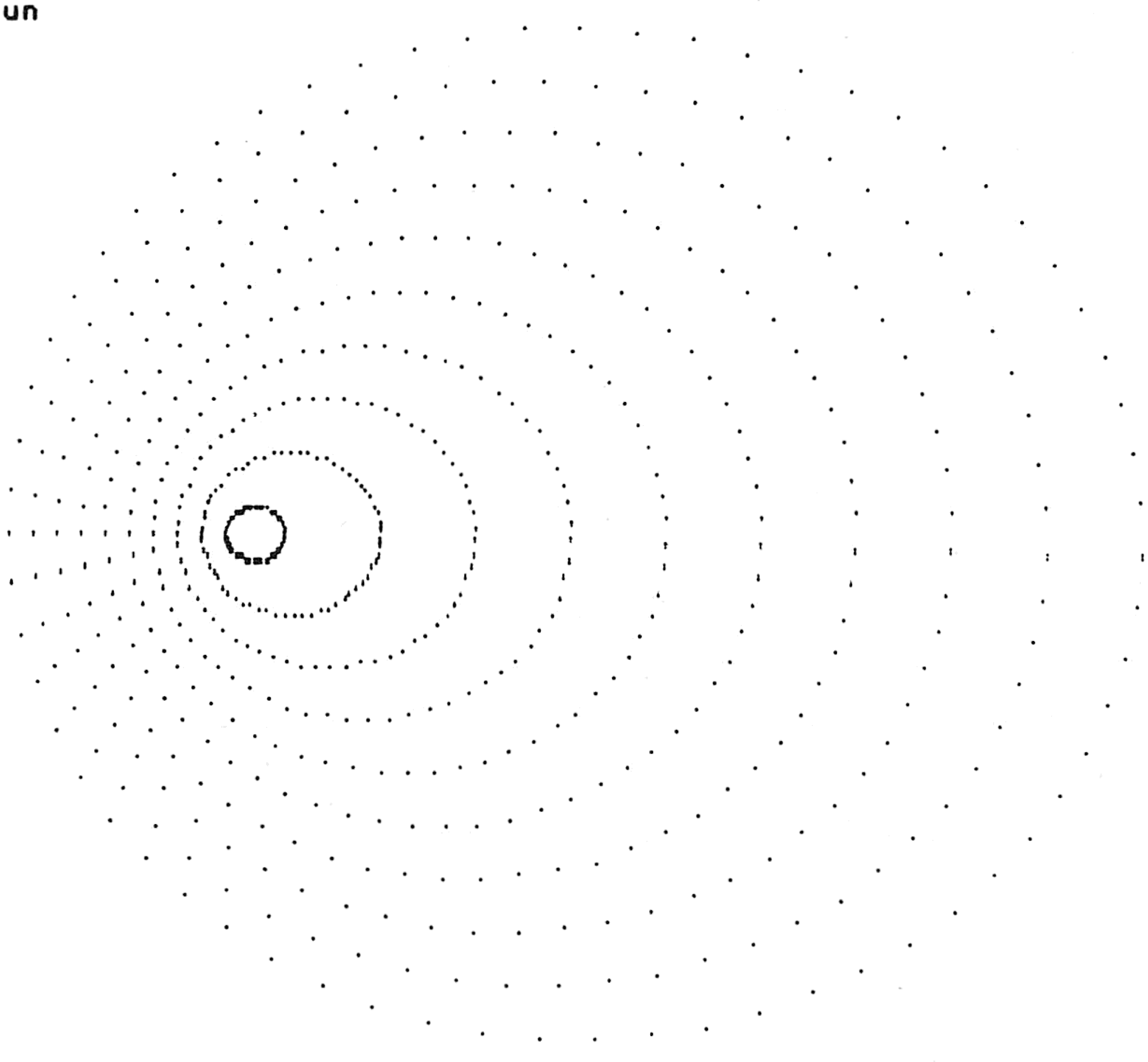
但し，N－BASIC では 1 つの FOR 文に対して，入れ子構造さえ正しくなっていれば，NEXT 文がいくつあってもかまいません。

注意：FOR/NEXT文の変数名としては，整数変数，および単精度変数のみが使えます。文字変数，倍精度変数は使用できません。

サンプルプログラム

```
100 SCREEN 0:CLS 3
110 FOR R=10 TO 200 STEP 20
120  FOR I=0 TO 3.14 STEP .05
130   Y=SIN(I)*COS(I)
140   X=SIN(I)*SIN(I)
150   Y=Y*R+100
160   X=X*R*2.5+100-C
170   PSET (X,Y)
180  NEXT I
190  C=C+10
200 NEXT R
210 END
```

```
run
```



FORMAT（ディスク）　N

機　能	ディスケットを初期化（フォーマット）する.
書　式	FORMAT <ドライブ番号>
文　例	FORMAT

解　説　フォーマットされていない新品のディスケット，または異なったシステムでフォーマットされたディスケット，または何らかの理由でデータを破壊してしまい，読み取り不能になったディスケットを再度使用するには，このFORMAT文を実行してディスケットをフォーマットしてください.

サンプルプログラム

```
format 2
format
```

GET　N88　N　（Nではディスクのみ）

機　能	ファイル中のデータをバッファに読み込む．
書　式	GET〔#〕〈ファイル番号〉〔，〈数〉〕
文　例	GET 3,5

解　説　〈ファイル番号〉で指定されたファイル中のデータを，対応するバッファ中に読み込みます．GETは，指定されたファイルがディスクファイルかどうかで，その動作が異なります．

〈ファイル番号〉で指定されたファイルがディスクファイルである場合には，GETは，ランダムファイルからの読み込みとして働きます．指定されたファイルは，ランダムアクセスのモードでオープンされていなければなりません．〈数〉はファイルのレコード番号として解釈され，指定されたレコード256バイトがバッファに読み込まれます．レコード番号が省略された場合には，直前に行われた，GET，PUTで指定されたレコードの次のレコードが読み込まれます．

GETを行った後で，INPUT #，LINE INPUT #等のシーケンシャル・アクセス入力を行った場合には，バッファに入っているデータから入力が行われますので注意してください．

〈ファイル番号〉で指定されたファイルがディスクファイルでない場合には，GETは，指定されたファイルに対応する装置から指定した文字数をバッファ中に読み込みます．指定されたファイルは入力モードでオープンされていなければなりません．〈数〉は，ファイルからバッファに読み込む文字数を意味し，0から255までの値で指定します．〈数〉が省略された時，及び0が指定された時には256文字を読み込みます．

GETはバッファへの読み込みを行うものです．変数にGETしたデータを割り付けるには，FIELD文を用います．

G

N-BASICでは，GETはディスクのランダムアクセスのみに用いることができます．

参照：OPEN，FIELD，PUT

サンプル
プログラム

```
100 '-- random data read --
110 OPEN "1:data1" AS #1
120 FIELD #1,30 AS A$,20 AS B$,50 AS C$
130 GET #1,1
140 PRINT "NAME",A$
150 PRINT "TEL",B$
160 PRINT "ADDRESS",C$
170 CLOSE

run
NAME            ｽｽﾞｷ
TEL             00-048-5423
ADDRESS         ﾁﾊﾞｹﾝ ﾅﾗｼﾉｼ 1-4-7
OK
```

GET@ N88

機　能　画面上のグラフィックパターンを読み込む．

書　式　GET〔@〕(Sx1，Sy1)－ | (Sx2，Sy2) / STEP(x, y) | ，〈配列変数名〉〔(〈要素〉)〕

文　例　GET(100, 50)－(130, 70), G%

解　説　画面上のスクリーン座標の2点（Sx1，Sy1）と（Sx2，Sy2）を結ぶ線を対角とする四角形の領域を，〈配列変数名〉で指定された配列変数に読み込みます．

気を付けなければならないのは，N88-BASICのグラフィック命令のほとんどがワールド座標を用いて使用するのに対し，このGET命令とPUT命令（PUT参照)は，ビューポート内に展開されるスクリーン座標（1章・ウィンドウとビューポート参照）を使用しなければならないことです．また2つ目の座標指定には相対座標による指定が許されています．

〈配列変数名〉は，ユーザーがグラフィックデータを読み込むために用意する数値型の配列変数の名前です．またこの配列変数に対して〈要素ナンバー〉を指定することもできます．この〈要素ナンバー〉は，グラフィックパターンを配列変数に読み込む際に，どの要素から格納し始めるかの指定です．省略した場合は配列の最初から格納します．1つの配列変数に対して複数のグラフィックデータを格納する場合に使うことができます．

但し，ユーザーはGET@命令に先だってこの配列変数を必要な大きさ確保しDIM文によって宣言しておかなければなりません．この配列変数の大きさは，読み込む領域の大きさ，現在のスクリーンモード，配列変数の型によって異なりますが，1つの配列に1つのパターンしか読み込まない場合は，次の計算式で求めることができます．複数パターンの場合はそれぞれのパターンについて計算し，その分確保しなければなりません．

〈必要なバイト数〉＝((〈横のドット数〉＋7)￥8)＊〈縦のドット数〉＊M＋4

　　但し　　白黒モードの時　――　M＝1

　　　　　　カラーモードの時　――　M＝3

〈添字の値〉＝〈必要なバイト数〉￥N＋1

　Nは配列変数の型によって変わります．

　　整数型配列　――　N＝2

単精度型配列　――　N＝4

倍精度型配列　――　N＝8

但しこの〈添字の値〉とは，一次元配列で添字の底が 0 の時（OPTION BASE 参照）の最小値です．一般に GET＠では一次元の配列変数を用います．

この GET＠文は後述する PUT＠文と密接な関係にあり，通常ペアで使用されます．

GET＠命令実行後，LP は (Sx_2, Sy_2)に移されます．

参照：PUT＠，OPTION BASE，1章・配列変数

サンプルプログラム

```
100 SCREEN 0,0:CLS 3
110 XD=40:YD=20
120 BYTE=((XD+7)¥8)*YD*3+4
130 FACT=BYTE¥2+1
140 DIM G%(FACT)
150 '
160 LINE(0,0)-STEP(XD-1,YD-1),1,B
170 CIRCLE(XD/2-1,YD/2-1),YD/2,4
180 GET(0,0)-STEP(XD-1,YD-1),G%
190 '
200 FOR X=0 TO 500 STEP 100
210 PUT(X,100),G%:NEXT
```

GET@　[N]

機能　画面上の任意の範囲のキャラクタまたは，ドットグラフィックスを配列へセーブします．

書式　1) GET〔@〕(X_1, Y_1)－(X_2, Y_2), <配列名>〔, G〕
　　　2) GET〔@〕A (X_1, Y_1)－(X_2, Y_2), <配列名>

文例　GET〔@〕(50, 50)－(100, 70), A%

解説　1) 任意の位置（X_1, Y_1）と（X_2, Y_2）を結ぶ線を対角線とする四角形の内側に表示されているキャラクタまたはドットグラフィックスを配列へセーブします．最後のオプションのGを付けるとドットグラフィックスをセーブすることになり，この場合の水平位置X_1, X_2は0から（1行の桁数）＊2－1までの値，垂直位置Y_1, Y_2は0から（画面の行数）＊4－1までの値をとります．Gを付けない場合はキャラクタをセーブすることになり，この場合のX_1, X_2は0から（1行の桁数）－1までの値，Y_1, Y_2は0から（画面の行数）－1までの値となります．<配列名>は整数型のものでプログラムの始めにDIM文で配列宣言をしておく必要があります．配列の大きさは一次元で，添字の最大値は指定した範囲に書きこめるキャラクタ数より大きな値を指定してください．

2) 書式1) の場合と異なりキャラクタとドットグラフィックスを一緒に配列へセーブすることができます．したがって，水平位置X_1, X_2は0から（1行の桁数）－1までの値，垂直位置Y_1, Y_2は0から（画面の行数）－1までの値をとります．さらにAを付けた場合には，カラーモードならその時点での色を，白黒モードならリバースやブリンクといった機能を同時に配列へセーブしています．その他の内容は1) の場合と同じです．

サンプルプログラム

```
100 CONSOLE 0,25,0,1:PRINT CHR$(12)
110 DIM G%(500)
120 LINE(10,10)-(40,20),"o",BF
130 LINE(30,50)-(70,70),PSET,BF
140 GET@(10,10)-(40,20),G%
```

GOSUB～RETURN　N88　N

機　能　サブルーチンのコール，及びサブルーチンからのリターンを行う．

書　式　GOSUB〈行番号〉
　　　～
　　RETURN〔〈行番号〉〕

文　例　GOSUB ＊GOOD

解　説　〈行番号〉から始まるサブルーチンプログラムをGOSUB文によってコールし，サブルーチンプログラム内のRETURN文によって，GOSUB文の次の文へリターンします．

サブルーチンプログラムとは，1つの他から独立したプログラムで，RETURN文で終っているものをいいます．これらは，GOSUB文によって，いつでも何度でもコールすることができます．

1つのサブルーチンの中から他のサブルーチンをコールすることもでき，これをサブルーチンの多重化と呼びます．この多重化はメモリのスタック領域の容量が許す限り行う事ができ，もし，スタック領域が足りなくなった場合には，〝Out of memory〟(OM Error) エラーが起こります．

サブルーチンは必ずGOSUB文によってコールされなければなりません．もし，単独で実行した場合，RETURN文に出会うと，〝RETURN without GOSUB〟(RG Error) エラーが起こります．

また，1つのサブルーチン内に複数のRETURN文があってもかまいませんが，正しくGOSUB文と対応していなければなりません．

なお，RETURN文では〈行番号〉を指定して，条件に応じて特定の行へ強制的にリターンさせることができます．

但し，N-BASICではRETURN文に〈行番号〉をつけることはできません．

注意：〝GO〟と〝SUB〟の間には最大1個のスペースを入れることが許されます。(もちろん，〝GO〟と〝SUB〟を続けて書いてもかまいません。)〝GO〟と〝SUB〟の間に2個以上のスペースを書いた場合，BASICはそれをGOSUB文とは解釈しません。

参照：CLEAR

サンプル
プログラム

```
100 ' サンカクケイ ノ メンセキ ヲ モトメル.
110 *START
120  INPUT "テイヘン :",TEIHEN
130  INPUT "タカサ  :",TAKASA
140  GOSUB *MENSEKI
150  PRINT "メンセキ ハ";MENSEKI
160  PRINT
170 GOTO *START
180 *MENSEKI
190  MENSEKI=TEIHEN*TAKASA/2
200 RETURN

run
テイヘン :100
タカサ  :50
メンセキ ハ 2500

テイヘン :145
タカサ  :20
メンセキ ハ 1450

テイヘン :10000
タカサ  :34
メンセキ ハ 170000

テイヘン :356
タカサ  :180
メンセキ ハ 32040

テイヘン :
Break in 120
Ok
```

GOTO／GO TO　N88 N

機能　指定した行へジャンプする.

書式　1）GOTO 〈行番号〉
　　　2）GO TO 〈行番号〉

文例　GOTO 500

解説　〈行番号〉で指定した行へジャンプします.

1), 2）は全く同機能です.

注意："GO" と"TO"の間のスペースは1個のみが許されます。2個以上のスペースを入れた場合，BASICはそれをGOTO文とは解釈しません。

サンプルプログラム

```
100 *START
110  PRINT "How do you do?"
120  GOTO *START

run
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
How do you do?
^C
Break in 110
Ok
```

HELP ON／OFF／STOP　N88

機　能	ヘルプキーによる割り込みの許可，禁止，停止を設定する。
書　式	1)　HELP ON 2)　HELP OFF 3)　HELP STOP
文　例	HELP OFF

解　説

1）割り込み動作を許可します。
この命令実行後は，ヘルプキーを押すごとに割り込みを発生し，ON HELP GOSUB によって定義された処理ルーチンに分岐します。

2）割り込み動作を禁止します。この命令実行後は，ヘルプキーを押しても処理ルーチンへの分岐は起こりません。

3）割り込み動作を停止します。この命令実行後は，ヘルプキーを押しても押されたことを覚えているだけで，処理ルーチンへの分岐は起こりません。
しかし，その後 HELP ON 命令によって割り込みが許可されると，先程のヘルプキーを押したことによって処理ルーチンに分岐します。

注意：プログラム終了時には HELP OFF を実行しておいて下さい。さもないと本来のヘルプキーの機能が失われます。

参照：ON HELP GOSUB，KEY ON／OFF／STOP

サンプルプログラム

```
HELP ON
HELP OFF
HELP STOP
```

H

IF…THEN…ELSE／IF…GOTO…ELSE　N88　N

機能　論理式の条件判断を行う.

書式

IF 〈論理式〉 | THEN | 〈文〉 | 〔ELSE | 〈文〉 | 〕
| | | 〈行番号〉 | | 〈行番号〉 |
| | GOTO 〈行番号〉 | | | |

文例　IF A$= ″y″　THEN ＊START ELSE 120

解説　論理式の条件によってプログラムの実行を制御します. 即ち, 〈論理式〉が真（0以外）ならば, THEN 或はGOTO文以降を実行し, 偽（0）ならばELSE文以降が実行されます. もし, ELSE文以降が省略されている場合には, 次の行が実行されます.

IF…THEN…ELSE文において, THEN文やELSE文に続けて別のIF文を置いて多重にすることができます. 但し, 多重できるのは1行に書ける範囲に限られます.

サンプルプログラム

```
100 '-- real or imaginary --
110 INPUT A
120 FLG=0
130  IF A<0 THEN A=ABS(A):FLG=1
140 PRINT SQR(A);
150  IF FLG THEN PRINT "i" ELSE PRINT
160 END

run
? 743
 27.258
Ok
run
? -653.23
 25.5584 i
Ok
run
? -3.9576
 1.98937 i
Ok
```

INIT% [N]

機　能	**RS-232C通信ポートを初期設定(イニシャライズ)します.**
書　式	**INIT %〈ポート番号〉, 〈モード〉, 〈コマンド〉**
文　例	**INIT %1, &HFB, &H37**

解　説　〈ポート番号〉で指定されたRS-232C通信ポートの入出力バッファのイニシャライズを行います. PC-8801MKII本体に内蔵されているRS-232Cインタフェースを使用する場合, 〈ポート番号〉は1を指定します.

〈モード〉と〈コマンド〉はプログラマブル・コミュニケーションインタフェースμPD8251Cの動作を指定します.

イニシャライズを行っていない状態でRS-232Cインタフェースを使用すると, “Port not initialized”エラーが起こります.

μPD8251Cの〈モード〉と〈コマンド〉は次のようにそれぞれ8ビットのデータを設定します.

〈モード〉

非同期モード

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S_2	S_1	EP	PEN	L_2	L_1	B_2	B_1

ボーレート

B_1	0	1	0	1
B_2	0	0	1	1
	同期モード	×1	×16	×64

キャラクタ長

L_1	0	1	0	1
L_2	0	0	1	1
	5ビット	6ビット	7ビット	8ビット

PEN → パリティ・イネーブル
1：イネーブル
0：ディスエイブル

EP → 偶数パリティ発生/チェック
1：偶数　0：奇数

ストップビット数

S_1	0	1	0	1
S_2	0	0	1	1
	無効	1ビット	1 1/2ビット	2ビット

同期モード

D7	D6	D5	D4	D3	D2	D1	D0
SCS	ESD	EP	PEN	L2	L1	0	0

キャラクタ長

0	1	0	1
0	0	1	1
5ビット	6ビット	7ビット	8ビット

(L1 → 上段, L2 → 中段)

PEN: パリティ・イネーブル
1：イネーブル
0：ディスエイブル

EP: 偶数パリティ発生/チェック
1：偶数　0：奇数

ESD: 外部同期検出
1：SYNDET＝入力
0：SYNDET＝出力

SCS: 単一キャラクタ同期
1：単一 SYNCキャラクタ
0：ダブルSYNCキャラクタ

〈コマンド〉

D7	D6	D5	D4	D3	D2	D1	D0
EH	IR	RTS	ER	SBRK	RXE	ETR	TXEN

TXEN: 送信イネーブル
1：イネーブル
0：ディスエイブル

ETR: データ・ターミナルレディ
1：Data Terminal ReadyをONにする
0：NO OPERATION

RXE: 受信イネーブル
1：イネーブル
0：ディスエイブル

SBRK: センドブレイク・キャラクタ
1：ブレイクキャラクタの送信
0：通常動作

ER: エラーリセット
1：エラーフラグ(PE, OE, FE)をリセット
0：NO OPERATION

RTS: センド要求
1：Request to SendをONにする
0：NO OPERATION

IR: 内部リセット
1：8251をモード・インストラクションフォーマットへもどす
0：NO OPERATION

EH: HUNT
1：SYNCキャラクタの検出を始める
0：NO OPERATION

サンプルプログラム

```
100 '-- init% sample --
110 'switch to RS-232C channel
120 OUT &H30,&H20 OR (&HF AND PEEK(&HEA66))
130 'jump table set
140 POKE &H8001,&H23:POKE &H8000,&H7F
150 'interrupt enable
160 OUT &HE6,4
170 'current states register set
180 OUT &HE4,&HFF
190 'ﾓｰﾄﾞﾊﾞｲﾄ :ｽﾄｯﾌﾟ ﾋﾞｯﾄ  2
200 '          :ｸﾞｳｽｳ ﾊﾟﾘﾃｨ
210 '          :ﾊﾟﾘﾃｨ ｲﾈｰﾌﾞﾙ
220 '          :ｷｬﾗｸﾀﾁｮｳ  7
230 '          :ﾎﾞｰﾚｲﾄ ﾌｧｸﾀ ＊16
240 'ｺﾏﾝﾄﾞﾊﾞｲﾄ:ｾﾝﾄﾞ ﾖｳｷｭｳ ON
250 '          :ｴﾗｰﾘｾｯﾄ
260 '          :ｼﾞｭｼﾝ ｲﾈｰﾌﾞﾙ
270 '          :ﾃﾞｰﾀ ﾀｰﾐﾅﾙ ﾚﾃﾞｨ
280 '          :ｿｳｼﾝ ｲﾈｰﾌﾞﾙ
290 INIT %1,&HFA,&H37
300 IF PORT(1) THEN PRINT INPUT$(PORT(1),%1);
310 C$=INKEY$:IF C$<>"" THEN PRINT %1,C$;
320 GOTO 300
330 '********************************
340 ' ﾌﾟﾛｸﾞﾗﾑ ｼｭｳﾘｮｳ ｺﾞ out &hE6,&h0
350 ' ｦ ｼﾞｯｺｳ ｼﾃ ｸﾀﾞｻｲ｡
360 '********************************
```

INPUT　N_{88}　N

機　能　指定した変数へ入力する.

書　式　INPUT〔〈〝プロンプト文″〉{ ; | , }〕〈変数〉〔,〈変数〉…〕

文　例　INPUT　〝Date″ ; A$

解　説　INPUT 文が実行されると，プログラムはキーボードからの入力待ちの状態となります．このとき，〈〝プロンプト文″〉にセミコロン（；）が続いた場合にはさらに疑問符（？）と1桁のスペースが画面に表示されますが，コンマ（,）が続いた場合には〈〝プロンプト文″〉の後には何も出力されません．

但し，N - BASIC の場合，INPUT 直後のセミコロン（；）と〈〝プロンプト文″〉直後のコンマは許されません．もし，使った場合には〝Syntax error″ となります．

データはリターンキーを押すことによって入力され，変数に代入されます．〈変数〉をコンマ（,）で区切って複数個指定した場合には，入力するデータもコンマで区切って〈変数〉の数だけ入力させなければなりません．また，対応する〈変数〉とデータの型とは一致していなければなりません．もし，個数や型が違っていた場合には〝? Redo from start″ を出力して再び入力待ちとなります．

また，何もせずにリターンキーを押した場合には 0 やヌルストリングが入力されたとみなされます．この場合も〈変数〉が複数の場合，必要数のコンマだけは入力しなければなりません．

N - BASIC の場合，〈変数〉が複数個指定されていた場合には，データをコンマで区切って同数入力してもかまいません．1つずつリターンキーを押して入力していってもかまいません．この場合，二回目からは〝??″ が表示されて入力待ち状態となります．また，データが〈変数〉の個数より多い場合には〝Extra ignored″ というメッセージを表示して次の文へ実行が移ります．

〈変数〉とデータの型が一致しなかった場合には，N_{88} - BASIC と同様に〝? Redo from start″ と表示し，最初から入力し直すことになります．

また，リターンキーだけを押した場合は，何も行われなかったと見なされ，〈変数〉の内容は INPUT 文を実行する直前と変わりません．複数個のデータ入力の途中でリターンキーだけを押した場合，残りすべての〈変数〉に対する入力がスキップされ，次の文へ実行が移ります．

文字変数に文字列を代入する場合，コンマや文字列の前後の意味のある空白を

入力したいときは，ダブルクォーツ（″）で文字列を囲む必要があります．この場合は文字として引用符を入力することはできません．これ以外の場合には，文字列を引用符で囲む必要はありません．しかし，その場合も引用符は文字列の最後の1文字にしか文字として使うことはできません．

サンプル
プログラム

```
100 INPUT "ﾅﾏｴ ";NA$
110 INPUT "ｾｲﾍﾞﾂ ";SEX$
120 INPUT "ﾈﾝﾚｲ ";OLD
130 PRINT "ﾅﾏｴ    :",NA$
140 PRINT "ｾｲﾍﾞﾂ :",SEX$
150 PRINT "ﾈﾝﾚｲ  :",OLD;"ｻｲ"

run
ﾅﾏｴ ? ﾀﾅｶ
ｾｲﾍﾞﾂ ? ｵﾄｺ
ﾈﾝﾚｲ ? ｶﾄｳ
?Redo from start
ﾈﾝﾚｲ ? 35
ﾅﾏｴ    :          ﾀﾅｶ
ｾｲﾍﾞﾂ  :          ｵﾄｺ
ﾈﾝﾚｲ   :           35 ｻｲ
Ok
```

INPUT #　N88 N （Nではディスクのみ）

機　能	シーケンシャルファイルからのデータを読み込む.
書　式	INPUT　#〈ファイル番号〉, 〈変数〉〔, 〈変数〉…〕
文　例	INPUT　#1, A, B

解　説　データを読み込む対象がシーケンシャルファイルであることと，疑問符（?）が出力されないことを除けば，INPUT 文とほぼ同じです.

〈ファイル番号〉は，OPEN 文によって，そのファイルを入力としてオープンしたときに使った番号です.

N - BASIC で〈ファイル番号〉に－1を指定すると，カセットからの入力となります.

INPUT #を実行し，もしバッファが空で，バッファ変更フラグが BASIC によってセットされていれば，まず，バッファの内容を書き出し，それから次のレコードをバッファに読み込みます.

ファイルに書き込まれているデータは，INPUT 文に対して必要なデータと同様の正しい形になっていなければなりません.

参照：OPEN

サンプルプログラム

```
100 '-- sequential data read --
110  OPEN "stest.dat" FOR INPUT AS #1
120 *SINPUT
130  IF EOF(1) THEN END
140  INPUT #1,A$
150  PRINT A$
160  GOTO *SINPUT
170 END

run
CAT
DOG
HORSE
SHEEP
MOUSE
OK
```

INPUT%　Ⓝ

機　能　**RS-232C通信ポートの入出力バッファからデータを入力します．**

書　式　**INPUT%〈ポート番号〉，〈変数名〉〔,〈変数名〉…〕**

文　例　**INPUT%1, A$**

解　説　〈ポート番号〉で指定されたRS-232C通信ポートの入出力バッファよりデータを入力し，その値を〈変数名〉で指定される変数に代入します。このときPC-8801MKII内蔵のRS-232Cインタフェースを使用する場合，〈ポート番号〉は1を指定します．

入力されるデータはCR，LFで区切られます．バッファの先頭からこの区切りをさがし，区切りが見つかったところで，それらを変数に代入します．もし，バッファ内にCR，LFがなければRS-232Cチャネルを通してこの区切りが入力されるまで待ちます．

参照：PRINT%，INIT%，PORT

サンプルプログラム

```
100 '-- RS-232C port input --
110 'switch to RS-232C channel
120 OUT &H30,&H20 OR (&HF AND PEEK(&HEA66))
130 POKE &H8001,&H23:POKE &H8000,&H7F
140 OUT &HE6,4
150 OUT &HE4,&HFF
160 INIT %1,&HFA,&H37
170 INPUT %1,A$
180 IF A$="end" THEN OUT &HE6,0:END
190 PRINT A$
200 GOTO 170
```

INPUT WAIT　N88

機　能	キーボードからの入力を時間制限する.
書　式	INPUT　WAIT 〈待ち時間〉, 〔〈"プロンプト文"〉 { ; / , } 〕〈変数名〉〔, 〈変数名〉…〕
文　例	INPUT　WAIT　100, "Your name" ; NA$

解　説　〈待ち時間〉で指定された時間だけキーボードからの入力を待ちます. 〈待ち時間〉 の単位は0.1秒です.

INPUT　WAIT は待ち時間に制限があることを除けば, INPUT 文と同様です. 但し, この文の後にマルチステートメントとして他の文を続けた場合, 制限時間内に入力が行われた時のみ後の文を実行し, 入力が行われなかった時は, 後の文を無視し次の行へ実行が移りますから, プログラミングの際は注意してください.

サンプルプログラム

```
100 '-- computer teacher --
110 *ENTRY:A=INT(RND*100):B=INT(RND*100)
120  PRINT "Ploblem",
130  PRINT A;"+";B"=";
140  INPUT WAIT 30,ANS:GOTO *EXIT
150  PRINT:PRINT "Time over !",:GOTO *ANS
160 *EXIT
170  IF ANS=A+B THEN PRINT "Right !":GOTO *ENTRY
180  PRINT "Wrong !",
190 *ANS:BEEP:PRINT "Answer =";A+B:GOTO *ENTRY

run
Ploblem         24 + 30 =? 54
Right !
Ploblem         31 + 51 =? 34
Wrong !        Answer = 82
Ploblem         5 + 78 =?
Time over !    Answer = 83
Ploblem         49 + 36 =?
Time over !    Answer = 85
Ploblem         98 + 90 =? 188
Right !
Ploblem         72 + 0 =?
Break in 140
Ok
```

KEY　N88 N

機　能	キーボードの上部にあるプログラマブル・ファンクションキーの内容を定義する.
書　式	KEY 〈キー番号〉, 〈文字列〉
文　例	KEY 1, "LOAD"

解　説　ファンクションキーは5つあり，シフトモードを加えると合計10の〈文字列〉を記憶させておくことができます．各ファンクションキーは最大15文字までの文字列およびコントロール文字をプログラムでき，キーボードから入力できない文字は，CHR$関数をプラス記号で継いで使います．〈キー番号〉は1から10までで，6から10まではシフトモードです．

サンプルプログラム

```
key list

load "              save "
auto                key
go to               print
list                edit .
run                 cont
Ok
key 3,"yes"+chr$(13)
Ok
key 8,chr$(&h22)+"BASIC"+chr$(&h22)
Ok
key list

load "              save "
auto                key
yes                 "BASIC"
list                edit .
run                 cont
Ok
```

KEY LIST　N88 N

機　能	ファンクションキーの内容をリストする.
書　式	KEY　LIST
文　例	KEY　LIST

解　説　ファンクションキーの内容の一部は画面の最終行に表示させておくことができますが，KEY　LIST では，そのすべての内容を画面に表示させることができます.

サンプルプログラム

```
key list

load "             save "
auto               key
go to              print
list               edit .
run                cont
Ok
```

KEY(n) ON／OFF／STOP　N88

機　能	ファンクションキーによる割り込みの許可，禁止，停止を設定する．
書　式	1）　KEY〔(〈キー番号〉)〕ON
	2）　KEY〔(〈キー番号〉)〕OFF
	3）　KEY〔(〈キー番号〉)〕STOP
文　例	KEY　ON

解　説　ファンクションキーが押された時に起こる割り込みを許可するか（ON），禁止するか（OFF），停止するか（STOP）を設定します．

〈キー番号〉とは，1から10までのプログラマブル・ファンクションキーの番号を表します．省略された場合は1から10までのすべてのファンクションキーに対して実行します．

1）割り込み動作を許可します．この命令実行後は，指定された〈キー番号〉のファンクションキーを押すごとに割り込みを発生し，ON KEY GOSUB によって定義された処理ルーチンに分岐します．

2）割り込み動作を禁止します．この命令実行後は，ファンクションキーを押しても処理ルーチンへの分岐は起こりません．

3）割り込み動作を停止します．この命令実行後は，ファンクションキーを押しても押されたことを覚えているだけで，処理ルーチンへの分岐は起こりません．しかし，その後 KEY(n) ON 命令によって割り込みが許可されると，先程のファンクションキーを押したことによって処理ルーチンに分岐します．

注意：ファンクションキーが ON の状態では，通常のプログラマブル・ファンクションキーとしては使用できません．プログラム終了時に KEY OFF を実行して下さい．またすべてのファンクションキーは，RUN 実行後 OFF の状態になります．

参照：ON KEY GOSUB

サンプルプログラム

```
key (1) on:key (3) on
key stop
key off
```

KILL（ディスク）　N88　N

機　能	ディスクからファイルを削除する．
書　式	KILL 〈ファイルディスクリプタ〉
文　例	KILL "2 : SAMPLE"

解　説　〈ファイルディスクリプタ〉で指定したファイルをディスクから削除します．

削除するファイルはクローズ状態になっていなければなりません．もしオープン状態のファイルを削除しようとすると "File already open" (AO Error) (ファイルは既に開かれている）エラーが起こります．また1つのKILL命令では1つのファイルしか削除できません．

KILL命令はすべてのディスクファイル(プログラム，ランダムデータ，シーケンシャルデータファイル）について使用できます．

注意：ドライブ番号は文例のようにファイル名の前に "n :" と指定します．省略された場合ドライブ1が指定されたとみなされます．

サンプルプログラム

```
100 OPEN "test2" FOR OUTPUT AS #1
110 PRINT #1,"KILL statement test"
120 CLOSE
130 '
140 FILES:PRINT
150 KILL "test2"
160 FILES

run
game          1     sample.     1     test2         1

game          1     sample.     1
OK
```

L

LET　N88 N

機　能	変数に値を代入する．
書　式	〔LET〕〈変数名〉＝〈式〉
文　例	LET　A＝10

解　説　キーワード LET はなくてもかまいません．つまり〈式〉の値を〈変数名〉に代入するには，等号だけでもよいのです．

〈式〉の内容は，数値，文字列のいかなる型であってもかまいません．但し，文字型から数値型への代入またはその逆は許されません．型の異なる数値型変数への代入では左辺の型に変換されます．

サンプルプログラム

```
100 LET A=123
110 LET B=.21
120 C=A+B
130 PRINT "A    =";A
140 PRINT "B    =";B
150 PRINT "A+B =";C
160 END

run
A    = 123
B    = .21
A+B = 123.21
Ok
```

LINE　N88

機　能　指定された２点間に直線を引く．

書　式　LINE〔 | (Wx_1, Wy_1) / STEP(x_1, y_1) | 〕－ | (Wx_2, Wy_2) / STEP(x_2, y_2) |
〔，〈パレット番号〉〕〔，| B / BF | 〕〔，〈ラインスタイル〉〕

文　例　LINE(100, 100)－(135, 100), 7, , &HF99F

解　説　ワールド座標系の２点（Wx_1，Wy_1）と（Wx_2，Wy_2）を結ぶ直線を描きます．(Wx_1，Wy_1) が省略された場合，LP の値を採用します．STEP を付けると相対座標による指定になります（１章・座標の指定の仕方参照）．

引く線の色は〈パレット番号〉により指定し，もしこれが省略された場合，現在 COLOR 文によって設定されている，グラフィック画面のフォアグラウンドカラーが採用されることになります．

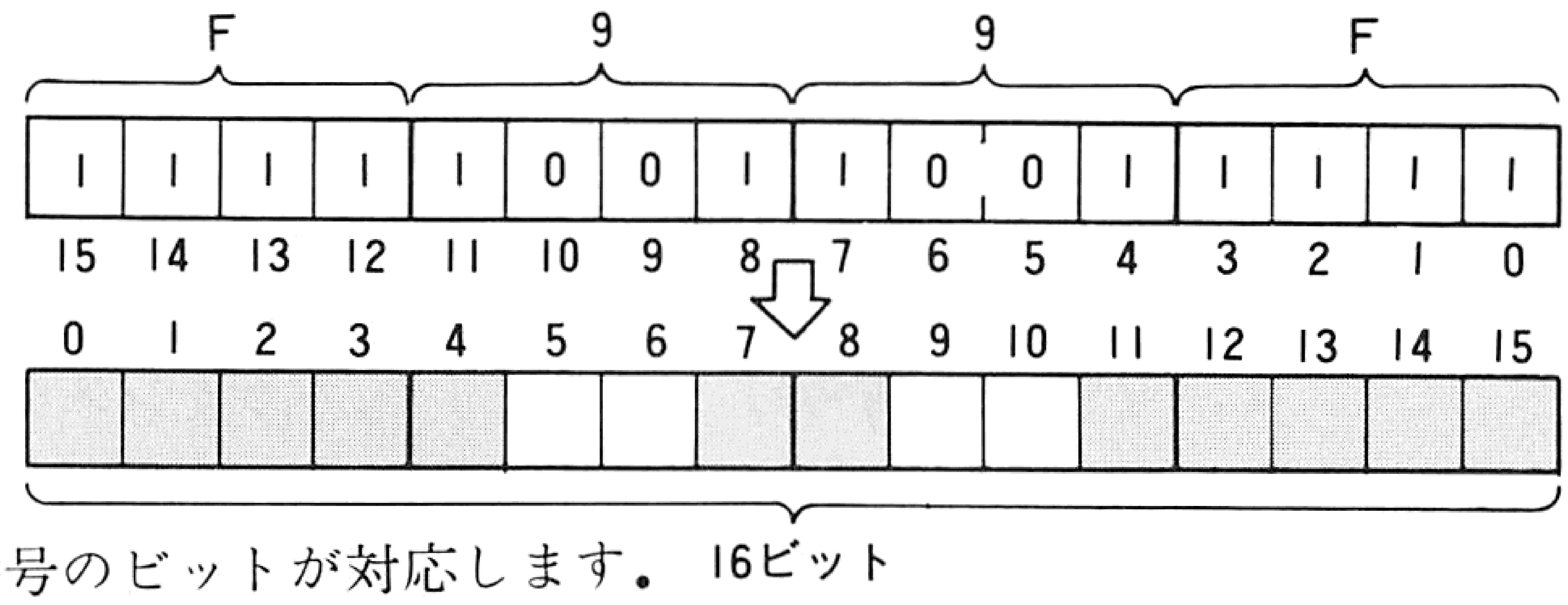
次のパラメータには，B か BF のどちらかを指定することができます．B は Box の意味で，(Wx_1，Wy_1) と (Wx_2，Wy_2) の２点を対角とする四角形の箱を描き，BF は Box Fill の意味でその四角形を塗りつぶします．但し〈ラインスタイル〉を指定した場合は，BF オプションを付けることはできません．

〈ラインスタイル〉は，引く線の型態を設定するオプションパラメータであり，16ビットで表すことのできる数，&H0 から&HFFFF（16進）の値をとることができます．この16進の数値とラインの型態との関係は次の図に示す通りで，16進の数の16ビットと画面上の16ビットが１対１に対応しており，〝1″の立っているビットに対応するドットが表示され，〝0″のビットに対応するドットは表示されません．この値は別に16進の数でなくてもよいのですが，16ビットに対応させて考える場合，16進数の方がわかりやすく一般的です．また線の長さを16ドット以上引いた場合，このラインスタイルが画面上の16ドットごとに繰り返されることになります．

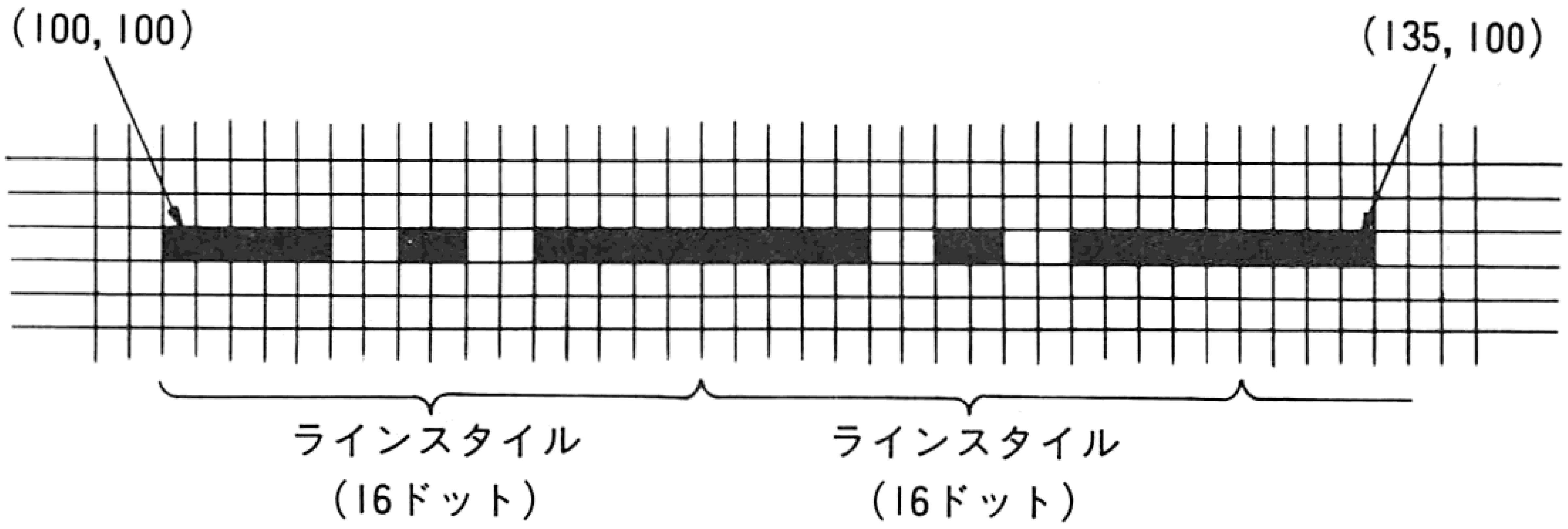
この図は，文例を実行した場合のものですが，水平座標の100から135までの36ドットの間に，１点鎖線のラインスタイルが繰り返されているのがわかります．

このようにラインスタイルを指定すると，16ドット単位で線の型態を決めることができますから，折れ線グラフや図面の作成の際，点線や１点鎖線など簡単に用いることができます．もし〈ラインスタイル〉が省略された場合，ノーマルな直線を引きます．

ラインスタイル（&HF99F）



※同じ番号のビットが対応します。

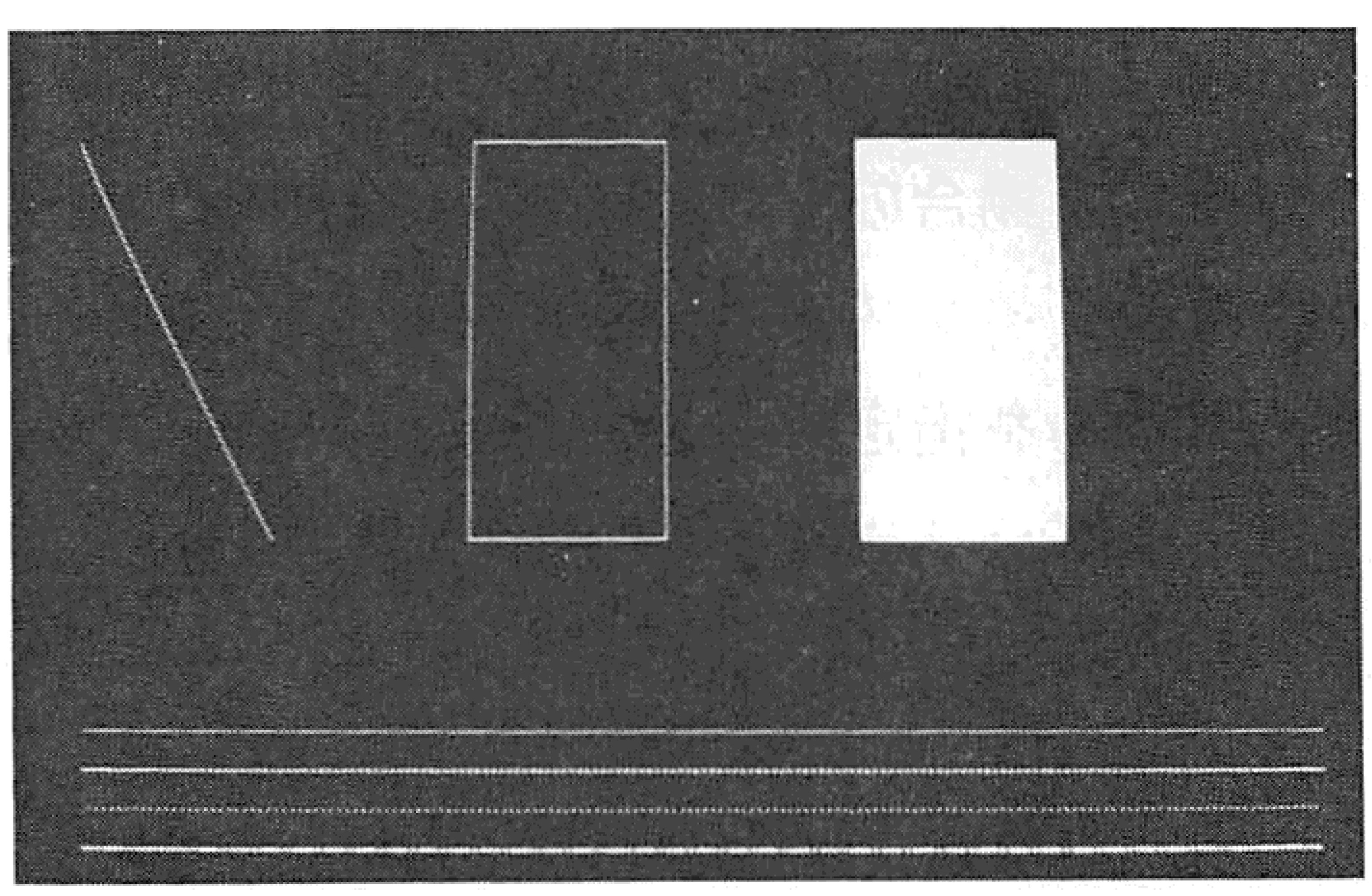


LINE(100, 100)－(135, 100), 7, , &HF99F を実行した時.

LINE 文を実行した場合，LP は $(Wx_2,\ Wy_2)$ に移動します.

サンプル
プログラム

```
100 LINE(0,0)-(100,100),1
110 LINE(200,0)-(300,100),2,B
120 LINE(400,0)-(500,100),3,BF
130 '
140 LINE(0,150)-STEP(639,0),4,,&HAAAA
150 LINE(0,160)-STEP(639,0),5,,&HCCCC
160 LINE(0,170)-STEP(639,0),6,,&H8888
170 LINE(0,180)-STEP(639,0),7,,&HF99F
```



LINE　Ⓝ

機　能　画面の行単位に機能を設定する，指定された2点間に線を引くなど．

書　式

1)　LINE 〈行指定〉，〈ファンクションコード〉

2)　LINE (X₁，Y₁)－(X₂，Y₂)，〈文字列〉
　　〔，〈ファンクションコード〉〔，| B / BF |〕〕

3)　LINE (x₁，y₁)－(x₂，y₂)，〈機能〉
　　〔，〈ファンクションコード〉〔，| B / BF |〕〕

文　例

1)　LINE 7，1

2)　LINE(0，0)－(79，24)，"♥"，B

3)　LINE(0，0)－(100，50)，PSET

解　説

1）カラーモードの画面において，行単位に機能を設定します．
〈行指定〉は0から（画面の行数）－1までの値をとり，〈ファンクションコード〉は次下の様になっています．

0	または	4	－	ノーマル
1	または	5	－	ブリンク
2	または	6	－	リバース
3	または	7	－	リバースブリンク

文例では，画面の8行目（即ち第7行）がブリンクします．

注意：この命令は白黒モードでは実行できません．

2）画面にキャラクタを使って，線を引いたり箱を書いたりします．
〈文字列〉の最初の文字列を使ってキャラクタ座標の2点 (X_1, Y_1)，(X_2, Y_2) を結ぶ直線を引きます．X_1，X_2 は水平位置を表し，0から（画面の桁数）－1の値，Y_1，Y_2 は垂直位置を表し、0から（画面の行数）－1の値をとります．ファンクションコードは，N-BASICのCOLOR文に用いられるものと同じです．
また最後にBを付けると (X_1, Y_1)，(X_2, Y_2) の2点を対角とする四角形の枠を描き，BFを付けるとその枠の内側もキャラクタで埋めます．

3）ドットグラフィックにより線や箱を書きます．

基本的な使い方は2)の場合と同じですが，キャラクタではなくドットで線を引きます．したがって与える座標はキャラクタ座標ではなくグラフィック座標です．

〈機能〉は，PSET または PRESET のどちらかで，PSET で線を書き PRESET で線を消します．

サンプルプログラム

```
100 CONSOLE ,,,1
110 LOCATE 0,0
120 PRINT "abcdefghijklmnopqrstuvwxyz"
130 LINE 0,1
```

```
100 CONSOLE ,,,1
110 LINE(0,0)-(10,10),"♥",2
120 LINE(1,1)-(9,9),"♠",5,BF
```

```
100 CONSOLE ,,,1
110 LINE(0,0)-(100,50),PSET,5
120 LINE(10,10)-(90,40),PSET,7,B
```

LINE INPUT　N88　N

機　能	1行全体（255文字以内）を区切ることなく文字変数に入力する．
書　式	LINE INPUT〔<"プロンプト文">；〕<文字変数>
文　例	LINE INPUT "NAME"；NA$

解　説　プロンプト文は，入力を受ける前にディスプレイにプリントされる文章です．疑問符（？）は表示されません．プロンプト以降キャリッジターンまでに入力されたすべての入力が<文字変数>に代入されます．

LINE INPUT 文の実行は CTRL-C またはストップキーの入力によって中断することができます．その場合，コマンドレベルに戻り，"Ok"とプリントします．

参照：LINE INPUT #，INPUT

サンプルプログラム

```
100 INPUT "INPUT : ";A$
110 LINE INPUT "LINE INPUT : ?";B$
120 PRINT:PRINT "INPUT : ",A$
130 PRINT "LINE INPUT : ",B$

run
INPUT : ? abcd,ABCD,1234
?Redo from start
INPUT : ? abcd
LINE INPUT : ?abcd,ABCD,1234,"HELLOW"

INPUT :         abcd
LINE INPUT :    abcd,ABCD,1234,"HELLOW"
Ok
```

LINE INPUT #　N88 N （Nではディスクのみ）

機能	1行全体（254文字以内）を区切ることなくシーケンシャル・ディスクファイルより文字変数に読み込む.
書式	LINE INPUT # <ファイル番号>, <文字変数>
文例	LINE INPUT #1, A$

解説　<ファイル番号>は, OPEN 文によって, そのファイルを入力としてオープンしたときに使った番号です. <文字変数>は, 行全体が代入される文字変数名です. LINE INPUT #文は, シーケンシャルファイルより, キャリッジリターンまでの, すべての文字を読みます. もし次に LINE INPUT #があればそのキャリッジリターンと続くラインフィードの組を飛ばして, 次のキャリッジリターンに至るまでのすべての文字を読みます. LINE INPUT #文は, データファイルのそれぞれの行がいくつかの欄に分割されているときや, アスキーモードでセーブしたプログラムを他のプログラムでデータとして読み込むときなどに特に有効です.

サンプルプログラム

```
100 OPEN "kybd:" FOR INPUT AS #1
110 LINE INPUT #1,A$
120 PRINT A$
130 CLOSE

run
1234567890,abcdefghijklmn,"opqrstuvwxyz"
Ok
```

LINE INPUT WAIT N88

機　能　キーボードからの入力を時間制限する.

書　式　LINE INPUT WAIT 〈待ち時間〉,〔〈"プロンプト文"〉 | ; | , |〕〈文字変数〉

文　例　LINE INPUT WAIT 50, "NO=", NO$

解　説　LINE INPUT WAIT は時間制限付きの LINE INPUT です．したがって LINE INPUT 文と INPUT WAIT 文の両方の機能を兼ねそなえています．詳しくはそれぞれの命令を参照してください．

参照：LINE INPUT, INPUT WAIT

サンプルプログラム

```
100 '-- type ﾉﾚﾝｼｭｳ --
110 *ENTRY
120 A$=""
130 FOR I=1 TO 3
140  A$=A$+CHR$(INT(RND*26+65))
150 NEXT I
160 PRINT "problem : ";A$
170 LINE INPUT WAIT 60,B$:GOTO 200
180 PRINT:PRINT "Time out!"
190 GOTO *ENTRY
200 IF A$=B$ THEN PRINT "Right!":GOTO *ENTRY
210 PRINT "Wrong!"
220 GOTO *ENTRY
run
problem : GHI
GH
Time out!
problem : NBU
NBU
Right!
problem : MJZ
MJX
Wrong!
problem : XSA

Break in 170
Ok
```

LIST/LLIST N88 N

機　能	メモリ内にあるプログラムの全部または一部をリストアウトする.
書　式	1）LIST〔〈行番号〉〕〔－〈行番号〉〕 2）LLIST〔〈行番号〉〕〔－〈行番号〉〕
文　例	LIST 100 - 200

解　説　プログラムの内容を，1）はディスプレイに2）はプリンタにリストアウトします. LIST コマンドは実行し終わると，いつでもコマンドレベルに戻ります.

〈行番号〉が省略された場合は，最も若い行番号のプログラム行よりリストが始まります（リストはプログラムの終り，またはストップキーの入力により終ります）. 最初の行番号のみが指定された場合には，その指定された行だけがリストされます. 最初の行番号とそれに続くハイフンまでが指定された場合は，その行番号に始まり，それよりも大きい行番号の行すべてがリストされます. ハイフンとそれに続く2番目の行番号だけが指定された場合には，プログラムの始めからその行番号までの行がリストされます. 両方の行番号がハイフンでつながれて指定された場合には，その範囲のすべての行がリストされます.

サンプルプログラム

```
list
100 '-- list test pogram --
110  N=1
120 *START
130  PRINT N
140  N=N+1
150  IF N<10 THEN *START
160 *LAST:END
Ok
list *start-*last
120 *START
130  PRINT N
140  N=N+1
150  IF N<10 THEN *START
160 *LAST:END
Ok
list .
160 *LAST:END
Ok
```

LOAD　N88　N　（Nではディスクのみ）

機　能	プログラムをメモリにロードする.
書　式	LOAD 〈ファイルディスクリプタ〉〔, R〕
文　例	LOAD "CAS2 : TEST"

解　説　〈ファイルディスクリプタ〉で指定されたプログラムファイルをメモリ上にロードします．ロードを実行すると，すべての開いているファイルを閉じ，変数はその値が失なわれますが，Rオプションをつけると，ファイルは開いたままで，プログラムをロード後，ただちに実行を開始します．

LOAD は，指定されたファイルを見つけて，実際のプログラムのロードを開始するまでは，メモリ中のプログラムを保存します．

N-BASICでは，LOAD コマンドは，ディスク版でのみ準備されており，ディスクファイルからのみ，プログラムをロードすることができます．

参照：SAVE, MERGE, CLOAD

サンプルプログラム

```
LOAD "cas1:sample1"
LOAD "1:smple2",R
LOAD "cas2:sample2"
```

LOAD？　N88 N　（Nではディスクのみ）

機能	カセットに正しくプログラムがセーブできたかを確認する．
書式	LOAD？ 〈ファイルディスクリプタ〉
文例	LOAD？ "CAS2：TEST"

解説　〈ファイルディスクリプタ〉により指定されたプログラムファイルの内容と，メモリ中のプログラムの内容が同一であるかを調べます．両者が異なっている場合には "Bad" と表示されます．

このコマンドは，通常SAVEの直後に，正しくセーブが行われたことを確認する目的で用います．またこのコマンドは，カセットファイルに対してのみ有効であり，他の装置上のファイルが指定された場合には通常のLOADコマンドと全く同じ動作をします．カセットファイルが指定された場合には比較を行うのみで実際のロード動作は行いませんから，メモリ中のプログラムは失なわれません．

サンプルプログラム

```
save "cas1:testpro"

load? "cas1:testpro"
```

LOCATE　N_{88}　N

機　能	テキスト画面のカーソルの位置を指定する.
書　式	LOCATE〔〈X〉〕〔,〈Y〉〕〔,〈カーソルスイッチ〉〕
文　例	LOCATE 10, 10

解　説　カーソルをテキスト画面のキャラクタ座標 (X, Y) の位置へ移動します.

〈X〉は水平座標を表し, 省略した場合は 0 とみなされます. 〈Y〉は垂直座標を表し, 省略された場合は現在カーソルがセットされている行とみなされます. これらは共に画面の左上を（0, 0）とするキャラクタ座標により指定します.（1章・ディスプレイ画面上の座標系参照）

〈カーソルスイッチ〉はカーソルの状態を決めるスイッチで, 0だと表示されなくなり, 1で表示されます.

注意： N-BASICでは〈X〉,〈Y〉を省略することはできません。また, プログラム実行時でも〈カーソルスイッチ〉の指定により, 表示するかどうかが選択できます.

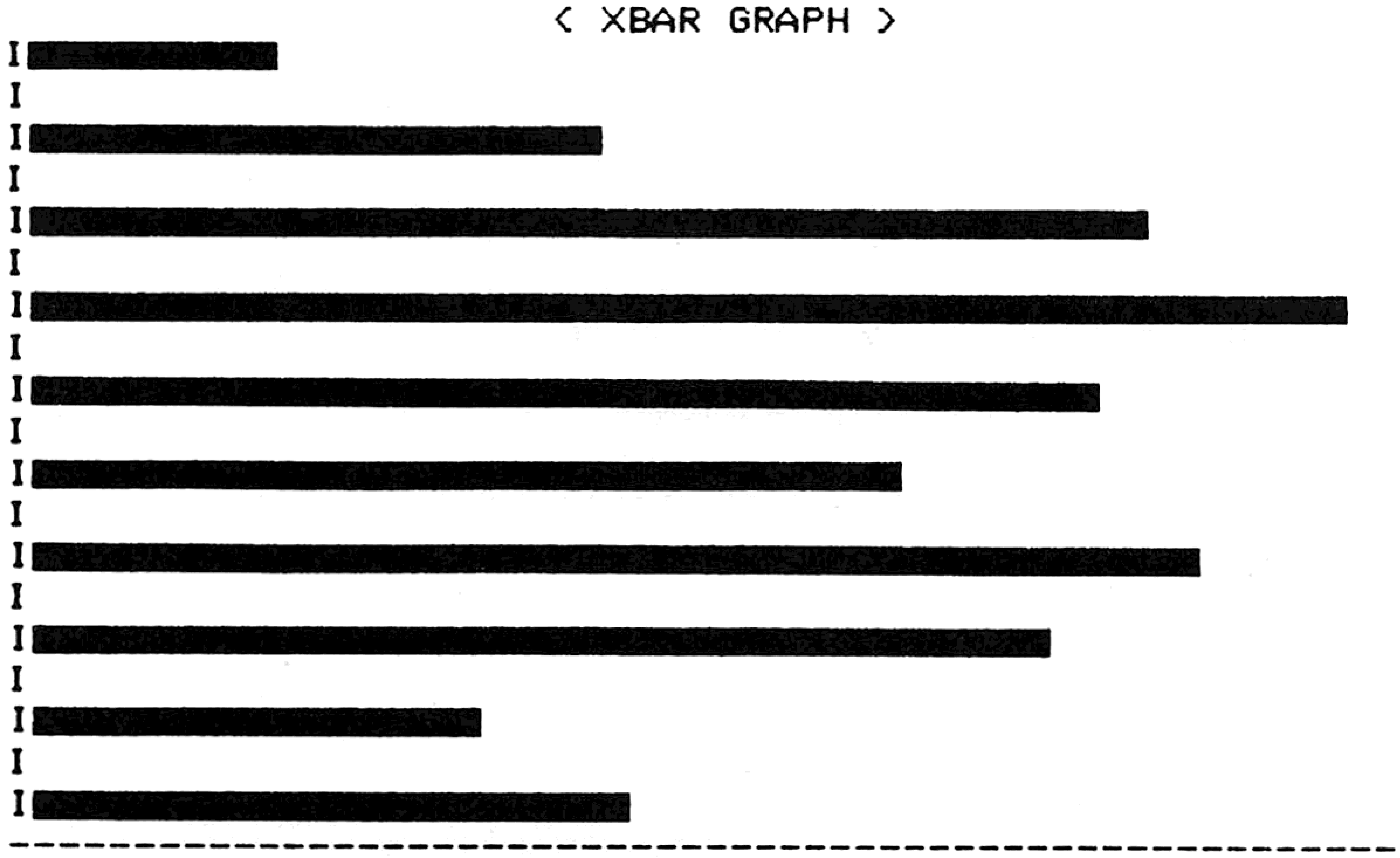
N_{88}-BASICでは, コマンドレベル(プログラムが走っていない状態)及びINPUT文など, キーボードからの入力待ちの場合にだけ〈カーソルスイッチ〉が有効となります.

サンプルプログラム

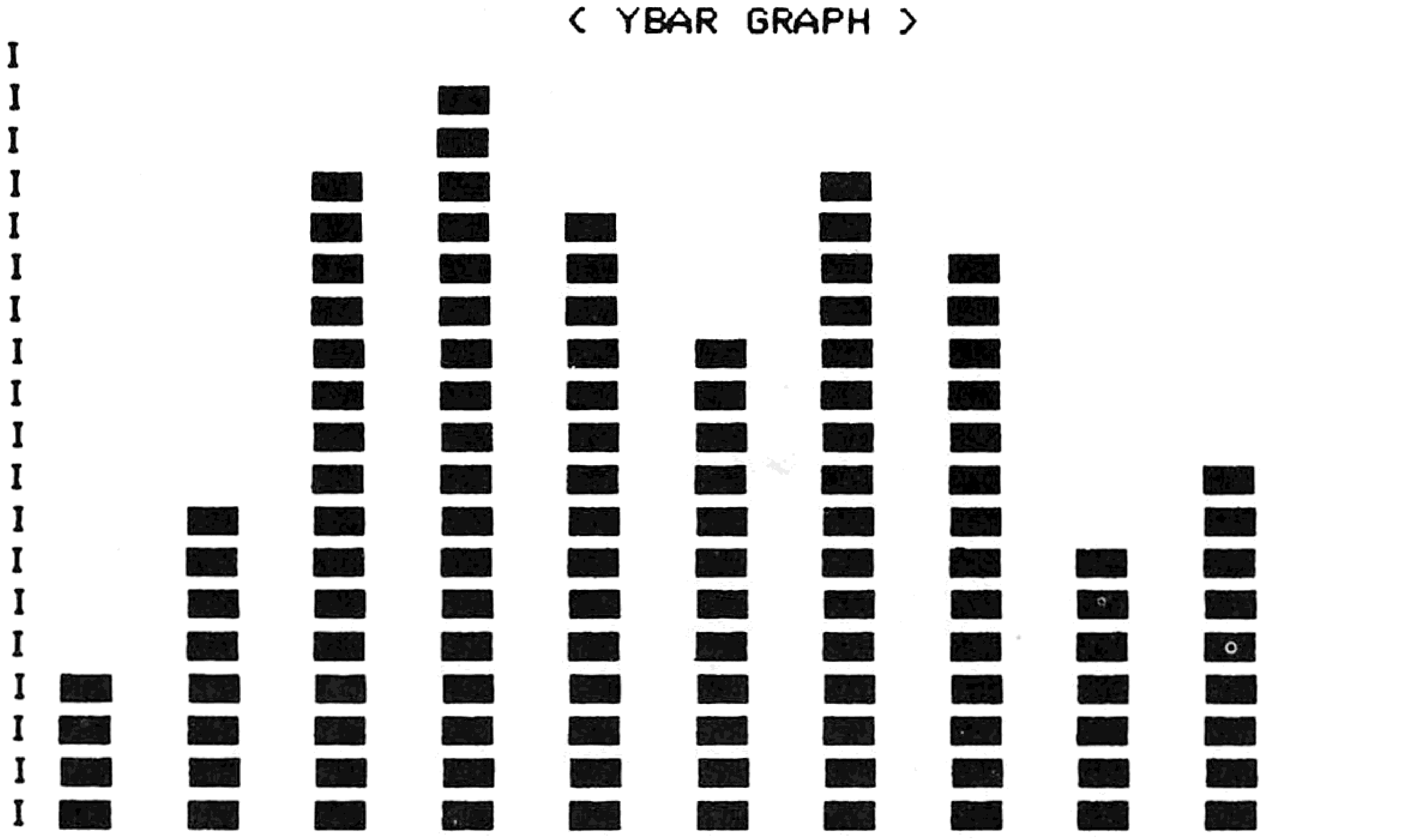
```
100 '-- character pattern --
110 CONSOLE 0,25,0,1:WIDTH 80,25:CLS
120 DATA 10,23,45,53,43,35,47,41,18,24
130 FOR I=1 TO 10:READ DAT(I):NEXT I
140 GOSUB *GRAPH:GOSUB *XBAR
150 A$=INPUT$(1):CLS
160 GOSUB *GRAPH:GOSUB *YBAR:END
170 *GRAPH:'-- graph draw --
180  X=3:FOR Y=1 TO 19
190   LOCATE X,Y:PRINT "I":NEXT Y
200  LOCATE 3,20:PRINT STRING$(56,"-"):RETURN
210 *XBAR:LOCATE 25,0:PRINT "< XBAR GRAPH >"
220  FOR I=1 TO 10
230   LOCATE 4,I*2-1:PRINT STRING$(DAT(I),CHR$(&H87))
240  NEXT I:RETURN
250 *YBAR:LOCATE 25,0:PRINT "< YBAR GRAPH >"
```

```
260  FOR I=1 TO 10
270   FOR J=19 TO 19-DAT(I)/3 STEP -1
280    LOCATE I*5,J:PRINT CHR$(&H87)
290    LOCATE I*5+1,J:PRINT CHR$(&H87)
300   NEXT J
310  NEXT I:RETURN
```

< XBAR GRAPH >



< YBAR GRAPH >



LSET／RSET　N88 N （Nではディスクのみ）

機　能	ファイルバッファへデータを書き込む.
書　式	LSET 〈文字型変数〉=〈文字列〉 RSET 〈文字型変数〉=〈文字列〉
文　例	LSET　A＄=〝PC-8801MK2〞 RSET　A$=〝N88 - BASIC〞

解　説　Left SET, Right SET の意味でランダム・アクセスファイルのレコードの, FIELD 文で定義した文字変数に該当する領域にデータを格納します. 格納するデータは文字型データでなければならず, 数値データは MKI$, MKS$, MKD$のいずれかにより文字形データに変換しなければなりません.

〈文字列〉の文字数が FIELD 文で割り当てられた長さより短い場合, LSET では左詰め, RSET では右詰めでフィールドを満たします. また, 〈文字列〉がフィールドより長い場合は, LSET, RSET とも右側の部分が失われます.

〈文字型変数〉は, FIELD 文であらかじめ定義されていなければなりません. 定義されていない変数を用いるとエラーが発生します.

参照：FIELD, PUT

サンプルプログラム

```
100 OPEN "1:rdata" AS #1
110 FIELD 1,20 AS A$,10 AS B$
120 LINE INPUT "ﾋﾝﾒｲ ?";NAM$
130 INPUT "ﾀﾝｶ ";PRICE%
140 LSET A$=NAM$
150 RSET B$=MKI$(PRICE%)
160 PUT #1,1
170 CLOSE

run
ﾋﾝﾒｲ ?ｴﾝﾋﾟﾂ
ﾀﾝｶ ? 100
Ok
```

MERGE（ディスク）　N88　N

M

機　能	プログラムを混合する.
書　式	MERGE <ファイルディスクリプタ>
文　例	MERGE "2 : TEST"

解　説　メモリ上のプログラムと<ファイルディスクリプタ>により指定したプログラムファイルを1つにして，メモリ上に置きます．ここで，指定されたファイルはアスキーセーブされていなければなりません．そうでない場合には，エラーになります．もし，ファイル中のプログラムと，メモリ中のプログラムに同一の行番号があった場合には，ファイル中の行でメモリ中の行を置き換えます．

MERGE コマンドは実行を終了すると，常にコマンドレベルに戻ります．

参照：SAVE

サンプルプログラム

```
merge "sample"

merge "2:testpro"
```

MID$　N88 N

機　能　文字列の一部を置き替える.

書　式　MID$ (〈文字列1〉, 〈式1〉〔, 〈式2〉〕) = 〈文字列2〉

文　例　MID$(A$, 3) = "ABC"

解　説　〈文字列1〉の〈式1〉番目の文字から〈式2〉個の文字を, 〈文字列2〉の最初から〈式2〉個の文字列で置き替えます.

〈式2〉が省略された場合, または〈文字列2〉の文字数より多く指定した場合は, 〈文字列2〉のすべての文字と置き替わります.

〈式1〉+〈式2〉−1の値が〈文字列1〉の文字数より大きくなり, 指定した文字列があふれてしまう場合は, (〈文字列1〉の文字数)−〈式1〉+1の値を〈式2〉とし, 〈文字列2〉の残りの部分は無視します.

また当然のことながら, 〈式1〉の値は〈文字列1〉の文字数より大きくてはいけないし, 0以下であってもいけません. したがって〈文字列1〉にヌルストリングを指定することはできません.

参照: 3章・MID$

サンプルプログラム

```
100 DATA My name is !!!!!!!!
110 DATA Sato,Tanaka,Suzuki,Murakami
120 READ A$
130 FOR I=1 TO 4
140  READ NAM$:MID$(A$,12)=NAM$
150  PRINT A$
160 NEXT I

run
My name is Sato!!!!
My name is Tanaka!!
My name is Suzuki!!
My name is Murakami
Ok
```

MON　N88　N

機　能　内蔵の機械語モニタに制御を移す．

書　式　MON

文　例　MON

解　説　BASICモードから内蔵の機械語モニタに制御を移すためのステートメントです．機械語モニタの命令はユーザーズ・マニュアルを参照してください．CTRL-Bで，もとのBASICモードにもどります．

サンプルプログラム

```
mon
```

MOTOR　N88 N

機　能	カセットテープレコーダのモータの ON，OFF を制御する.
書　式	MOTOR〔〈スイッチ〉〕
文　例	MOTOR 1

解　説　〈スイッチ〉を 0 にすればモータは OFF となり，0 以外の値にすればON となります．また，〈スイッチ〉を指定しない場合，モータが ON の状態ならばOFF に，OFF の状態ならばON にします．

サンプルプログラム

```
100 MOTOR 1
110 PRINT "Rewind and set the tape."
120 INPUT "Are you ready (y/n):",A$
130 IF LEFT$(A$,1)<>"y" THEN END
140 PRINT
150 MOTOR 0
160 PRINT "Push PLAY button."
170 INPUT "Are you ready (y/n):",A$
180 IF LEFT$(A$,1)<>"y" THEN END
190 MOTOR
200 END

run
Rewind and set the tape.
Are you ready (y/n):y

Push PLAY button.
Are you ready (y/n):y
Ok
```

MOUNT（ディスク） N

機　能	ディスクのファイル配置表を読んで以後のディスク入出力を可能にする。
書　式	MOUNT〔<ドライブ番号>〔, <ドライブ番号> …〕〕
文　例	MOUNT

解　説　ディスクとの入出力（OPEN, PRINT #, DSKI$など）を行う前にMOUNT命令を実行しなければなりません。

<ドライブ番号>を省略した場合には，現在接続されているすべてのドライブがマウントされます。MOUNT命令を実行するとBASICはディスケットからファイルの配置表を読み込み，誤りを調べます。誤りがなければ，そのディスケットはマウントされます。もし誤りがあった場合には，BASICはディスクをマウントするために予備の配置表を読み込もうとします。その場合次のメッセージがプリントされます。〝X copies of allocation bad on drive Y〃 Xは1または2，Yはドライブ番号です。この警告が起きた場合は，ファイルの配置表に何らかのトラブルが発生している訳ですから，新しくコピーを作ることをお勧めします。

もし予備の配置表のコピーが悪ければ，致命的なエラーである〝Bad allocation table〃となり，そのディスケットはマウントされません。

ディスケットがマウントされている間に，BASICは時折そのディスケットに配置表を書き込みますが，そのドライブにリード・アフターライトのアトリビュートがセットされていなければ，エラーチェックを行いません（SET文参照）。

N_{88}-BASICではオートマチックMOUNT, REMOVEを採用しているため，これらの操作はすべてBASICが管理します。

参照：REMOVE, SET

サンプルプログラム

```
mount 1
mount 2
```

NAME (ディスク)　N88　N

N

機　能	ディスクファイルの名前を変える.
書　式	NAME <旧ファイル名> AS <新ファイル名>
文　例	NAME "SAMPLE" AS "SAMPLE 1"

解　説　<旧ファイル名>で表わされているディスク上のファイルを<新ファイル名>に変更します. ドライブの指定はファイルディスクリプタ中にドライブ番号を含めることによって行います. ドライブ番号を省略した場合, ドライブ1が指定されたとみなされます.

なおファイル名の変更を行うファイルはクローズされた状態でなければなりません.

参照 : KILL

サンプルプログラム

```
100 OPEN "oldtest" FOR OUTPUT AS #1
110 FOR J=1 TO 10
120  WRITE #1,J
130 NEXT J:CLOSE
140 FILES:PRINT
150 NAME "oldtest" AS "newtest"
160 FILES

run
game         1     sample.      1     oldtes t     1

game         1     sample.      1     newtes t     1
Ok
```

NEW　N88　N

機能　メモリにあるプログラムを抹消し，すべての変数をクリアする．

書式　NEW

文例　NEW

解説　NEWコマンドは，コマンドレベルにあるとき新しいプログラムを入力する前に実行します．NEWコマンドの実行が終わると，いつもコマンドレベルに戻ります．

NEWコマンドは，開かれているファイルがある場合は，それを自動的に全て閉じます．

サンプルプログラム

```
100 ' NEW ｺﾏﾝﾄﾞ ﾉ ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ
110 '
120 ' NEW ｺﾏﾝﾄﾞ ﾊ ﾌﾟﾛｸﾞﾗﾑ ｦ
130 ' ｲﾆｼｬﾗｲｽﾞ ｼﾏｽ
140 '
150 PRINT " NEW ｺﾏﾝﾄﾞ ﾊ ﾌﾟﾛｸﾞﾗﾑ ｦ"
160 PRINT " ｲﾆｼｬﾗｲｽﾞ ｼﾏｽ."
170 END

run
 NEW ｺﾏﾝﾄﾞ ﾊ ﾌﾟﾛｸﾞﾗﾑ ｦ
 ｲﾆｼｬﾗｲｽﾞ ｼﾏｽ.
Ok
new
Ok
list
Ok
```

NEW ON [N88]

機　能　N88-BASICからN-BASICを起動するなど，PC-8801MKIIの種々のモードを設定する．

書　式　NEW ON 〈式〉

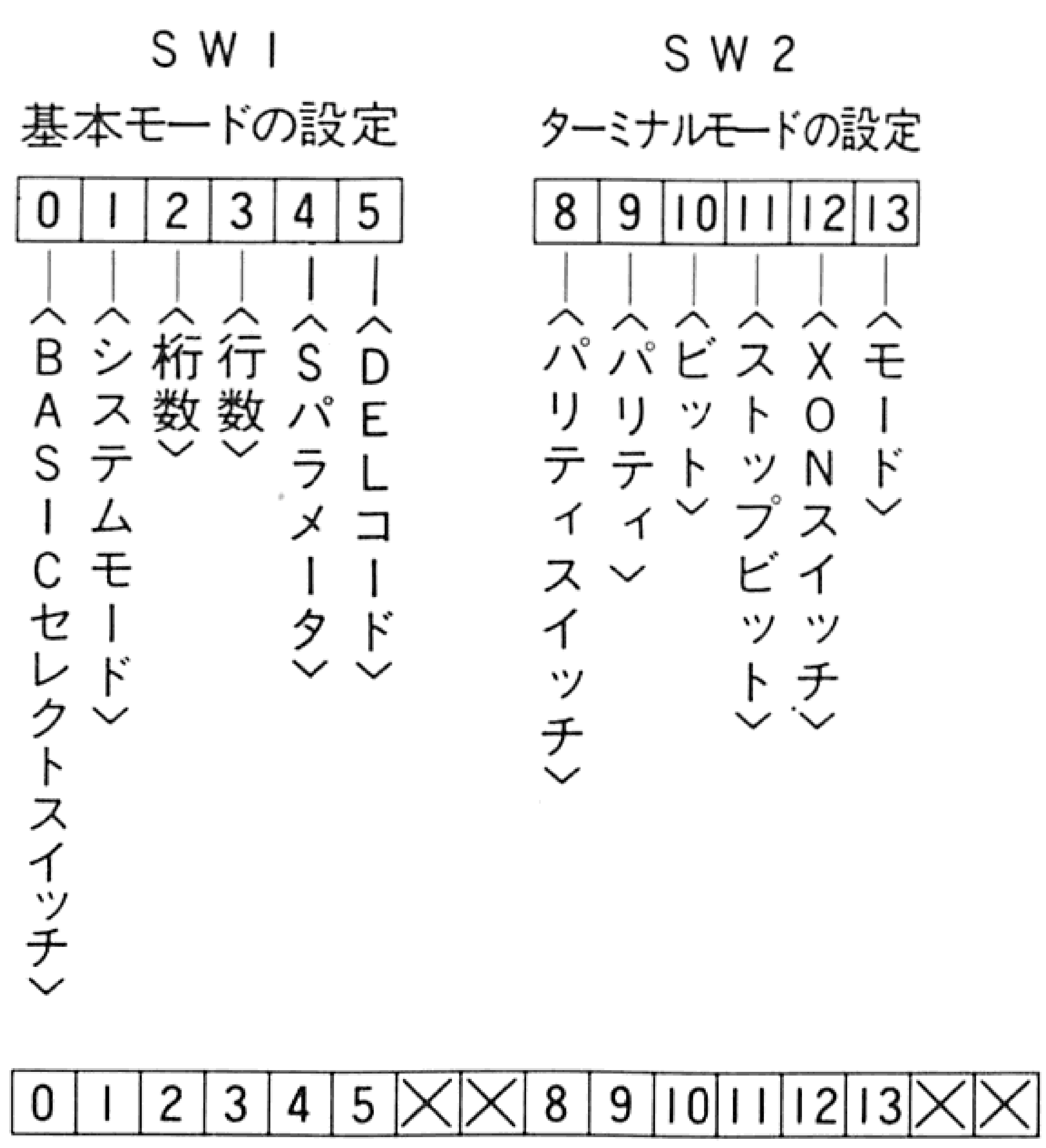
文　例　NEW ON 1

解　説　PC-8801MKⅡをN88-BASICモードにするかN-BASICモードにするか，またはターミナルモードにするかなどの最も根本的なモードの選択は，電源投入時のディップスイッチの設定によって決めることができますが，このNEW ON文は，それらのモードをソフトウェアによって変える機能があります。

〈式〉の値は，ディップスイッチの1個が1ビットに対応した，16ビット表現の値を指定します．PC-8801MKⅡには，16個のディップスイッチがありますが，その内NEW ON文で指定できるのは，SW1の1～6，SW2の1～6の計12個です．残りの4個(6，7，14，15ビット)は意味を持たないビットとなります．

例えばN88-BASICからN-BASICへ移行するだけならNEW ON 1を指定します．

ディップスイッチの機能



〈BASICセレクトスイッチ〉	0：N88-BASIC
	1：N-BASIC
〈システムモード〉	0：BASICモード
	1：ターミナルモード
〈桁数〉	1：1行80桁
	0：1行40桁
〈行数〉	1：1画面25行
	0：1画面20行
〈DELコード〉	0：DELコードを無視する。
	1：DELコードを処理する。
〈パリティスイッチ〉	0：パリティOFF
	1：パリティON
〈パリティ〉	0：偶数パリティ
	1：奇数パリティ
〈ビット〉	0：7ビット
	1：8ビット
〈ストップビット〉	0：ストップビット1.5
	1：ストップビット1
〈XONスイッチ〉	0：OFF
	1：ON
〈モード〉	0：フルデュプレックス
	1：ハーフデュプレックス
〈Sパラメータ〉	0：OFF
	1：ON

サンプルプログラム

```
new on 1
```

ON COM GOSUB　N88

機　能　通信回線からの割り込みが発生した時，分岐する処理ルーチンの開始行を定義する．

書　式　ON COM GOSUB <行番号>

文　例　ON COM GOSUB 1000

解　説　通信回線からRS-232Cのコミュニケーションポートに入力割り込みがあった時，分岐する処理ルーチンの開始行を定義します．

<行番号>とは分岐させる処理ルーチンの開始番号です．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じでRETURN命令によって行います．単にRETURNとした時は，中断した所から再開し，後ろに行番号を指定した時は，その行から再開します．

注意：この命令によって割り込み処理ルーチンに分岐させるには，割り込みが発生した時，COM ONの状態でなければなりません．

1つのON COM GOSUB文にラベルと行番号そのものを混在させることはできません．

参照：COM ON／OFF／STOP，OPEN，TERM

O

サンプルプログラム

```
100 ON COM GOSUB *COM.TRAP
110 OPEN "COM1:E72NN" AS #1
120 COM ON
130 PRINT TIME$: GOTO 130
140 END
150 '
160 *COM.TRAP
170 IF LOC(1)=0 THEN RETURN
180 PRINT INPUT$(LOC(1),1);
190 RETURN
```

ON ERROR GOTO　N88　N

機　　能	エラートラップを可能にし，エラー処理ルーチンの開始行を定義する．
書　　式	ON ERROR GOTO 〈行番号〉
文　　例	ON ERROR GOTO 1000

解　　説　エラートラップ機能が可能になると，エラーが検出された時指定されたエラー処理ルーチンへプログラムの制御が移ります．もし〈行番号〉の行が存在しなければ，"Undefined line"（未定義行）(UL Error)エラーが起こります．エラートラップ機能を禁止するには，ON ERROR GOTO 0 を実行してください．その後のエラーはエラーメッセージを表示し，プログラムの実行は停止します．また，エラー処理ルーチンに ON ERROR GOTO 0 の文がある場合には，BASIC はエラートラップの原因となったエラーのエラーメッセージを表示し，プログラムの実行を停止します．すべてのエラー処理ルーチンは，エラー回復処理を行わないエラーをトラップした場合には，ON ERROR GOTO 0 を実行することをお勧めします．

注意：エラー処理サブルーチンの実行中にエラーが起きた場合には，それに対するエラーメッセージが表示され，実行は停止します．エラー処理サブルーチンの中ではエラートラップは起こりません．

参照：RESUME

サンプルプログラム

```
100 ' X ノ y ジョウ ヲ モトメル
110 ON ERROR GOTO *ERRORMES
120 *START:INPUT "X :";X:INPUT "y :";Y
130   IF X=0 THEN ERROR 250
140  Z=X^Y: PRINT "  y":PRINT " X =";Z
150 GOTO *START
160 *ERRORMES
170  IF ERR=250 THEN PRINT " テイギ サレマセン."
180  IF ERR=6 THEN PRINT " オーバー フロー デス."
190 RESUME *START

run
X :? 2
y :? 10
```

```
   y
 X  =  1024
X  :? 100
y  :? 100
 オーバ゛ー フロー デ゛ス.
X  :? 0
y  :? -2
 テイキ゛ サレマセン.
X  :? 2
y  :? 8
   y
 X  =  256
X  :?
Break in 120
Ok
```

ON…GOSUB／ON…GOTO　N88　N

機　　能　指定されたいくつかの行に分岐する.

書　　式　ON 〈式〉 GOSUB 〈行番号〉〔, 〈行番号〉 …〕
ON 〈式〉 GOTO 〈行番号〉〔, 〈行番号〉 …〕

文　　例　ON A GOSUB 100, 200, 300, 400
ON A GOTO 100, 200, 300, 400

解　　説　〈式〉の値がプログラムのどの行番号の行に分岐するかを決定します. 〈行番号〉の並びは1から始まる数に対応します. たとえば, 〈式〉の値が3であったなら, 行番号のリストの3番目の行が分岐先となります.

ON…GOSUB文では, リストの各行番号はサブルーチンの最初の行の行番号でなければなりません.

〈式〉の値が負になった場合には, "Illegal function call" (FC Error) エラーが起こりますが, 値が0またはリストの行番号の個数より大きくなった場合には, 次の行へプログラムの制御が移り, エラーは起こりません.

注意：1つのON…GOSUB/ON…GOTO文にラベルと行番号そのものを混在させることはできません.

サンプルプログラム

```
100 *ENTRY
110  INPUT "N =";N
120  SG=SGN(N)+1
130  ON SG GOTO *ZERO,*PLUS
140  PRINT "MINUS!":GOTO *ENTRY
150 *ZERO:PRINT "ZERO!":GOTO *ENTRY
160 *PLUS:PRINT "PLUS!"
170  GOTO *ENTRY

run
N =? 12
PLUS!
N =? 45
PLUS!
N =? 0
ZERO!
N =? -45
MINUS!
N =? -1.45
MINUS!
N =? 10000
PLUS!
N =?
```

ON HELP GOSUB　N88

機　能	ヘルプキーによる割り込みルーチンの開始行を定義する.
書　式	ON HELP GOSUB <行番号>
文　例	ON HELP GOSUB 1000

解　説　ヘルプキーを押した時に，割り込みによって分岐する処理ルーチンの開始行を定義します.

ON HELP GOSUBはファンクションキーによる割り込みの1つであり，特殊な用途のためのON KEY GOSUBだと考えることができます．したがって，その使い方も同じです. 但し，ファンクションキーの場合は，INPUT文などキー入力の状態での割り込みは禁止されていますが，ヘルプキーの割り込みはそれが可能です.

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じでRETURN命令によって行います. 単にRETURNとした時は中断した所から再開し，後ろに行番号を指定した時は，その行から再開します. INPUT文実行中で割り込んだ場合は，単にRETURNで復帰させるとキー入力がなされる前にINPUT文の次の文から実行を再開しますから，行番号を指定するか，プログラムで適切な処置をしなければなりません（APPENDIX・割り込みキーの使い方参照).

本来この命令は，給与計算などのアプリケーション・プログラムを作成した際，プログラマーが，そのプログラムの使い方，入力の仕方などを説明したインストラクション・ルーチンを定義しておき，ユーザーがそのプログラム実行中に，次に何をしてよいのかわからなくなった時にヘルプキーによる割り込みで指示を受ける，といったような用途のために用意されたものです.

注意：この命令によって割り込み処理ルーチンに分岐させるには，割り込みがあった時HELP ONの状態でなければなりません.

参照：HELP ON／OFF／STOP, ON KEY GOSUB

サンプル
プログラム

```
100 '-- help key ｦ ﾂｶｯﾀ message print --
110  ON HELP GOSUB *MESSAGE
120  HELP ON
130 '
140  FOR I=1 TO 10
150   INPUT N
160   PRINT "N =";N,"N*N =";N*N
170  NEXT I
180  HELP OFF:END
190 '-- instruction --
200 *MESSAGE
210  HELP OFF
220  PRINT
230  PRINT "ｽｳｼﾞｦ INPUT ｼﾃｸﾀﾞｻｲ｡ "
240  PRINT "ｿﾉ 2 ｼﾞｮｳｦ ｹｲｻﾝｼﾏｽ｡"
250  HELP ON
260 RETURN 140

run
? abc
?Redo from start
?
ｽｳｼﾞｦ INPUT ｼﾃｸﾀﾞｻｲ｡
ｿﾉ 2 ｼﾞｮｳｦ ｹｲｻﾝｼﾏｽ｡
? 10
N = 10          N*N = 100
? 20
N = 20          N*N = 400
? 12.45
N = 12.45       N*N = 155.003
? 10,23
?Redo from start
?
ｽｳｼﾞｦ INPUT ｼﾃｸﾀﾞｻｲ｡
ｿﾉ 2 ｼﾞｮｳｦ ｹｲｻﾝｼﾏｽ｡
?
Break in 150
Ok
```

ON KEY GOSUB N88

機　能	ファンクションキーによる割り込みルーチンの開始行を定義する.
書　式	ON KEY GOSUB <行番号>〔, <行番号> …〕
文　例	ON KEY GOSUB 100, 200, 300, 400

解　説　ファンクションキーを押した時に分岐する処理ルーチンの開始行を定義します.

この命令は, 割り込みが発生した時（キーが押された時）にどの処理ルーチンに飛ぶかを定義するだけのものであり, この命令が実行されても割り込みが起こることはありません. 割り込みが発生するのは, あくまで何らかの外的要因（この場合, キーが押された時）があった時です.

<行番号>とは, 割り込みが発生した時に分岐させる処理ルーチンの開始行番号で, この並びの順序は, ファンクションキーのキー番号と1対1に対応していなければなりません. PC-8801MKIIがサポートしているファンクションキーは全部で10番までありますから, 最大10個の<行番号>を並べて書くことができます. もちろん実際の行番号の代わりにラベル名を使うこともできます.

処理ルーチンからの復帰は, 一般のサブルーチンの場合と同じでRETURN命令によって行います. 単にRETURNとした時は, 中断した所から再開し, 後ろに行番号を指定した時は, その行から再開します.

注意：この命令によって割り込み処理ルーチンに分岐させるには, 割り込みが発生した時, KEY ONの状態でなければなりません.

1つのON KEY GOSUB文にラベルと行番号そのものを混在させることはできません.

参照：KEY ON／OFF／STOP

サンプル
プログラム

```
100 '-- sin , cos --
110  ON KEY GOSUB *SINSUB,*COSSUB,*ENDSUB
120  FOR I=1 TO 4
130   KEY(I)ON
140  NEXT I
150  DEG=0:PI=3.14159
160 *START
170   TH=DEG/180*PI
180   PRINT "TH =";DEG;
190   IF FLAG=1 THEN *COSC
200   PRINT TAB(12);"SIN(TH) =";SIN(TH):GOTO *EXIT1
210 *COSC:PRINT TAB(35);"COS(TH) =";COS(TH)
220 *EXIT1:DEG=DEG+10
230  GOTO *START
240 '-- key interrupt routine --
250 *SINSUB:FLAG=0:RETURN
260 *COSSUB:FLAG=1:RETURN
270 *ENDSUB:KEY OFF:END
```

```
run
TH = 0      SIN(TH) = 0
TH = 10     SIN(TH) = .173648
TH = 20     SIN(TH) = .34202
TH = 30     SIN(TH) = .5
TH = 40     SIN(TH) = .642787
TH = 50                               COS(TH) = .642788
TH = 60                               COS(TH) = .500001
TH = 70                               COS(TH) = .342021
TH = 80                               COS(TH) = .173649
TH = 90                               COS(TH) = 1.49803E-06
TH = 100                              COS(TH) =-.173647
TH = 110                              COS(TH) =-.342019
TH = 120    SIN(TH) = .866027
TH = 130    SIN(TH) = .766046
TH = 140    SIN(TH) = .642789
TH = 150    SIN(TH) = .500002
TH = 160    SIN(TH) = .342023
Ok
```

ON PEN GOSUB　N88

機　能	ライトペンによる割り込み処理ルーチンの開始行を定義する.
書　式	ON PEN GOSUB <行番号>
文　例	ON PEN GOSUB ＊PENTEST

解　説　ライトペンが押されて割り込みが起こった時，分岐する処理ルーチンの開始行を定義します.

この命令によって処理ルーチンに分岐させるには，PEN ON の状態でなければなりません.

この命令は対象がライトペンであることを除けば，他の割り込み定義命令（KEY，COM，HELP，STOP）と同様です.

参照：ON KEY GOSUB etc.，PEN ON／OFF／STOP

サンプルプログラム

```
100 COLOR 0,7:CLS 2
110 ON PEN GOSUB *PENSUB
120 PEN ON
130 GOTO 130
140 '-- pen routine --
150 *PENSUB
160 PRINT "triger =";PEN(0),"x =";PEN(1),"y =";PEN(2)
170 RETURN

run
triger = 0     x = 39         y = 10
triger = 0     x = 29         y = 12
triger = 0     x = 23         y = 14
triger = 0     x = 8          y = 12
triger = 0     x = 28         y = 6
triger = 0     x = 59         y = 5
triger = 0     x = 59         y = 5
triger = 0     x = 58         y = 5
^C
Break in 130
Ok
```

ON STOP GOSUB　N88

機　能	ストップキーによる割り込み処理ルーチンの開始行を定義する.
書　式	ON STOP GOSUB <行番号>
文　例	ON STOP GOSUB 100

解　説　ストップキーを押した時に，割り込みによって分岐する処理ルーチンの開始行を定義します.

この命令は割り込みの対象としてストップキーを使っていることを除けば，ファンクションキーによる割り込みと同じですから，ON KEY GOSUB の項を参照してください.

この命令はアプリケーションプログラムの実行中に，ユーザーが誤ってストップキーを押してしまい，プログラムの実行が中断されてしまうのを防止するためのものです．したがって不用意にこの命令を実行してしまうと本来のストップキー及びCTRL-Cの機能は全く失われ，プログラムを中断することができなくなってしまいます．動作ミスのないプログラムにしてから使用してください.

参照：ON KEY GOSUB, STOP ON／OFF／STOP

サンプルプログラム

```
100 ON STOP GOSUB *MASK
110 STOP ON
120 F$="###     ##.######^^^^  ##.######^^^^"
130 PRINT "Degree";TAB(12);"Sin";TAB(27);"Cos"
140 FOR I=0 TO 180 STEP 20
150  TH=I/180*3.14159
160  PRINT USING F$;I;SIN(TH);COS(TH)
170 NEXT I:STOP OFF:END
180 *MASK:RETURN

run
Degree      Sin            Cos
  0      0.000000E+00   1.000000E+00
 20      3.420200E-01   9.396930E-01
 40      6.427870E-01   7.660450E-01
 60      8.660250E-01   5.000010E-01
 80      9.848080E-01   1.736490E-01
100      9.848080E-01  -1.736470E-01
120      8.660270E-01  -4.999980E-01
140      6.427890E-01  -7.660430E-01
160      3.420230E-01  -9.396920E-01
180      2.808800E-06  -1.000000E+00
Ok
```

ON TIME$ GOSUB　N88

機　能　リアルタイムタイマの割り込み発生時刻と，その時分岐する処理ルーチンの開始行を定義する.

書　式　ON TIME$＝"hh : mm : ss" GOSUB 〈行番号〉

文　例　ON TIME$＝"07 : 30 : 00" GOSUB ＊MORNINGCALL

解　説　割り込み時刻の設定の仕方はTIME$の場合と同様です．但し，ここで設定する時刻はリアルタイムタイマに影響を与えません．またこの命令で制御できる分解能は2秒ですから，場合によっては±1秒程度の誤差がでることがあります．

〈行番号〉は，割り込みによって分岐する処理ルーチンの開始行です．もちろんラベル名を使うこともできます．

ON TIME$ GOSUBは実行された時点で，設定された時刻から現在の時刻を引き算して割り込み時刻を決定します．したがって，もし現在の時刻をTIME$文で変更してから割り込み時刻を設定する場合は，TIME$文は必ずON TIME$ GOSUB文の直前で実行しなければなりません．

またこの命令によって割り込み処理ルーチンに分岐させるには，TIME$ ONの状態でなければなりません．

その他，処理ルーチンからの復帰などについては，ON KEY GOSUB文を参照してください．

注意：19：00：00以降に割り込み時間を設定した場合，割り込みがかからない場合があります．

参照：TIME$ ON／OFF／STOP，ON KEY GOSUB，TIME$

サンプルプログラム

```
100 ON TIME$="07:30:00" GOSUB *MORNING
110 TIME$ ON
120 LOCATE 30,10:PRINT TIME$:GOTO 120
130 '-- Morning call routine --
140 *MORNING
150 FOR I=1 TO 10
160  BEEP:PRINT "Good Morning !!"
170  FOR J=0 TO 300:NEXT J
180 NEXT I:END
```

OPEN　N88　N　(Nではディスクのみ)

機　能	ファイルを開く.
書　式	OPEN <ファイルディスクリプタ>〔FOR <モード>〕AS 〔#〕<ファイル番号>
文　例	OPEN "CAS1 : TEST" FOR INPUT AS #1

解　説　<ファイルディスクリプタ>で指定されたファイルを，指定された<ファイル番号> でオープンします．以後，オープンされたファイルへの入出力は，<ファイル番号> を指示することにより行います．

<モード>は，ファイルへのアクセス方法を指定します．モードには次の4種類があります．

INPUT	既存のファイルから入力を行うことを指示します．
OUTPUT	新しくファイルを作り出力を行うことを指示します．
APPEND	既存のファイルの終りから追加を行うことを指示します．
省略した時	FOR 節が省略されると，ランダムアクセスを行うことを指示します．

INPUT, APPEND モードでは，指定されたファイルが存在しないと，"File not found" (FF Error) のエラーとなります．OUTPUT モードでは，常に指定された名前のファイルを新しく作り，同一名のファイルがあった場合にはそのファイルは削除されます．ランダムアクセスでファイルが存在しない場合には新たに作られます．

<ファイル番号> は，1から15までの値を用いることができますが，BASIC を使い始める時に指定したファイルの数を越えてはなりません．また，既にオープンされているファイルが利用している番号を用いることはできません．

OPEN は，以後の入出力の際に用いるバッファ領域を確保し，ファイルが閉じられるまでの間，そのバッファは，指定したファイルへの入出力操作専用に用いられます．このバッファはファイル番号と同一の番号で参照されます．

RS-232C コミュニケーション・ファイルをオープンする際には，パリティ，ビット数等を指定する必要があります．これらは TERM コマンドと共通ですのでそちらを参照してください．

COM1：と CAS1：, CAS2：はハードウェア上の共通のポートを使用するために同時にオープンすることはできません．

また，カセットに対しては APPEND モードの指定及び，指定の省略はできま

せん。RS-232C コミュニケーション・ファイルは<モードの指定>を省略すれば，送信・受信の両方に使用できます。

N-BASIC では，OPEN はディスクファイルに対してのみ準備されており，他の装置上のファイルをオープンすることはできません。

参照：CLOSE, VARPTR, FIELD, TERM

サンプルプログラム

```
100 '-- ﾃﾞｰﾀ ﾃﾝｿｳ ﾌﾟﾛｸﾞﾗﾑ --
110 '      DSK1: --> DSK2
120 '
130  OPEN "1:data3" FOR INPUT AS #1
140  OPEN "2:newdata3" FOR OUTPUT AS #2
150 '
160 *START
170  IF EOF(1) THEN *EXIT
180    INPUT #1,A$
190    WRITE #2,A$
200  GOTO *START
210 *EXIT
220  CLOSE
230  END
```

OPTION BASE　N88

機　能	配列の添字の下限を宣言する.
書　式	OPTION BASE ｜0｜ 　　　　　　　　　｜1｜
文　例	OPTION BASE 1

解　説　配列の添字の下限を 0 または 1 にすることを宣言します．通常，この宣言が用いられない場合には，下限は 0 ですが，OPTION BASE 1 を実行すると下限は 1 になり，以後，配列の添字として 0 が用いられると，"Subscript out of range" (BS Error : Bad Subscript) エラーが起こるようになります．

この宣言は，プログラム中で配列変数が宣言，または引用された後に行うことはできません．また一度宣言すると，再宣言によって下限を変更することはできません．このような場合には "Redimensioned array" (DD Error : Duplicate Definition) エラーになります．

OPTION BASE は一度宣言すると，RUN あるいは CLEAR を行うまでは，解除，変更することができません．

サンプルプログラム

```
100 OPTION BASE 0
110 DIM A(1)
120 PRINT "A(0) =";A(0),"A(1) =";A(1)
130 '
140 CLEAR:OPTION BASE 1
150 '-- error occur --
160 PRINT "A(0) =";A(0),"A(1) =";A(1)

run
A(0) = 0        A(1) = 0
A(0) =
Subscript out of range in 160
Ok
```

OUT　N88　N

機能	コンピュータの出力ポートに1バイトのデータを送る.
書式	OUT <ポートアドレス>, <式>
文例	OUT 64, 32

解説　<ポートアドレス>はポートの番号, そして<式>が出力するデータです. これらは共に0から255までの整数で指定します.

サンプルプログラム

```
100 DEFINT A-Z
110 SCREEN 0,0:CLS 3
120 FOR X=80 TO 639 STEP 80
130  C=X/80
140  CIRCLE (X,100),50,C
150  PAINT (X,100),C,C
160 NEXT X
170 FOR C1=1 TO 21
180  FOR C2=0 TO 6
190   C=((C1+C2) MOD 7)+1
200   IF C>8 THEN C=C-8
210 ' OUT &H54+i,c ﾊ COLOR=(i,c) ﾄ ｵﾅｼﾞ
220   OUT &H55+C2,C
230  NEXT C2
240  FOR W=0 TO 100
250  NEXT W
260 NEXT C1
270 END
```

〔1〕PAINT　N88

機　能	指定された境界色で囲まれた領域を，指定された色で塗る．
書　式	PAINT {(Wx, Wy) | STEP(x, y)} 〔，〈領域色〉〔，〈境界色〉〕〕
文　例	PAINT(−50, 120), 3, 4

解　説　(Wx, Wy) により指定されたワールド座標を含む〈境界色〉で囲まれた領域を，指定された〈領域色〉で塗りつぶします．STEP を付けると相対座標となります（1章・座標の指定仕方参照）．

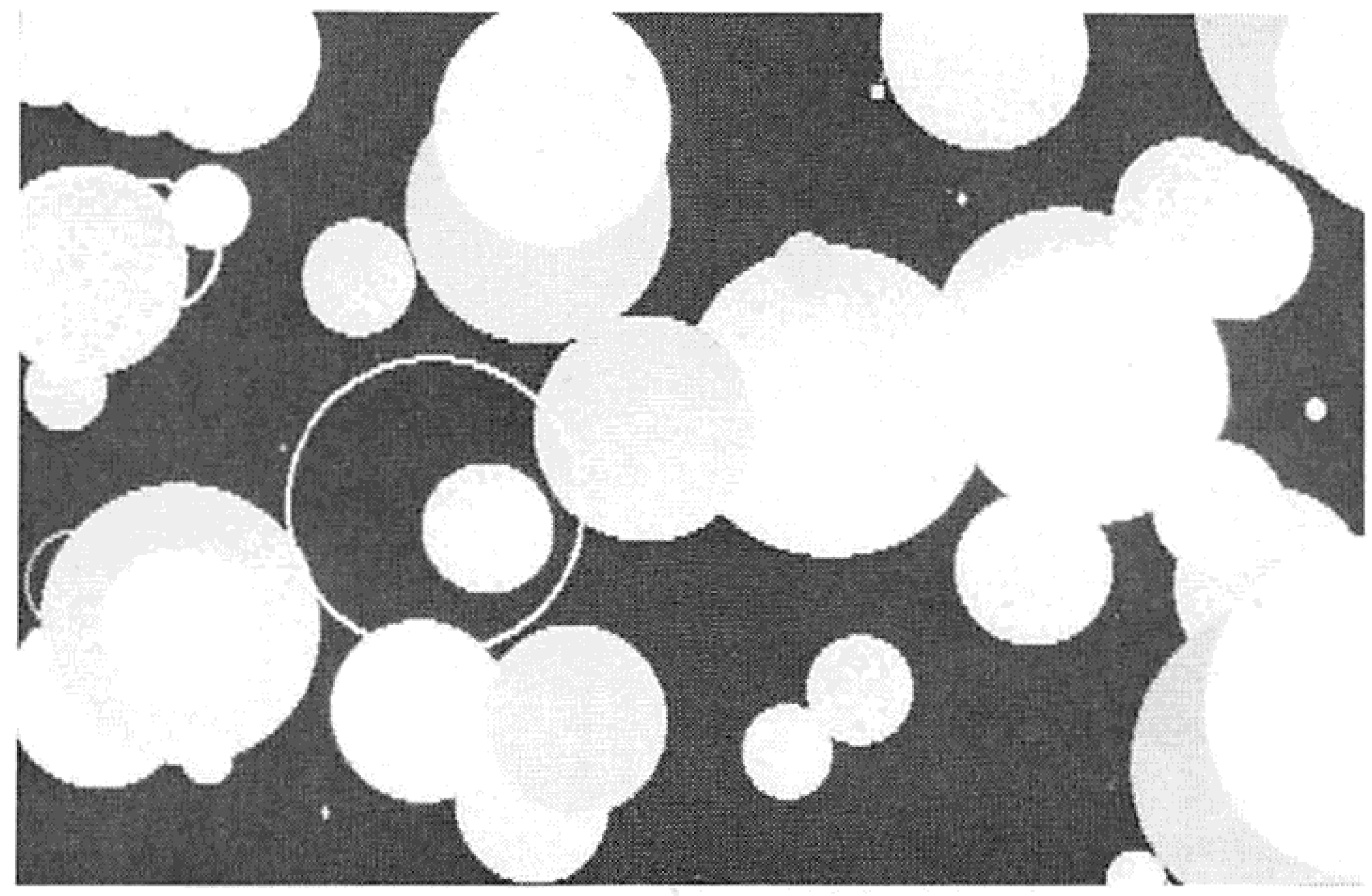
〈領域色〉，〈境界色〉はともにパレット番号で指定を行います．〈領域色〉が省略された場合には，COLOR 文で指定されたフォアグラウンドカラーが用いられます．〈境界色〉が省略された場合には〈領域色〉の指定と同じパレット番号が〈境界色〉として用いられます．

(Wx, Wy)は塗り始める座標を指定します．もしこの点が，既に指定された境界色と同じ色であった場合には，PAINT は何の画面操作も行いません．PAINT はビューポート内でのみ働きますので，ビューポートの境界は PAINT 動作の境界とみなされます．また，(Wx, Wy)がウィンドウの外にある場合には，対応するスクリーン座標はビューポート外になるために，"Illegal function call" (FC Error) エラーとなります．

参照：VIEW，COLOR〔1〕

サンプルプログラム

```
100 '-- normal paint --
110 SCREEN 0,0
120 CLS 3
130 FOR I=1 TO 50
140  X=RND*639
150  Y=RND*199
160  R=RND*80
170  C=INT(RND*7+1)
180  CIRCLE(X,Y),R,C
190  PAINT(X,Y),C
200 NEXT I
```



〔2〕PAINT　N88

機　能	指定された境界色で囲まれた領域を，指定されたタイルパターンで埋める．
書　式	PAINT {(Wx, Wy) / STEP(x, y)} ,<タイルストリング>〔,<境界色>〕〔,<バックグラウンド・ストリング>〕
文　例	PAINT(255, 120), TILE$, 4, BACK$

解　説　(Wx, Wy) により指定されたワールド座標を含む，〈境界色〉で囲まれた領域を，〈タイルストリング〉により指定された模様(タイル)で埋めます．STEPを付けると相対座標による指定となります．

この動作はタイリングと呼ばれ，模様のついたタイルを領域内に敷きつめたり，中間色で領域内を塗ったような効果を出すことができます．

〈タイルストリング〉は，タイリングに用いられる基本タイルの模様と大きさを決定する文字列です．タイルの大きさは，横方向は8ドット分と決められていますが，縦方向の長さは，タイルストリングの長さで指定することができます．縦方向がnドットのタイルを指定するためには，白黒モードでn文字，カラーモードで3・n文字の長さを必要とします．

タイルの模様は，〈タイルストリング〉に対応するキャラクタコードの2進数表現（ビットパターン）により表現されます．〈タイルストリング〉の指定の方法はカラーモードと白黒モードとで異なります．白黒モードでは，文字を横8ビットの線に対応させて指定します．文字コードは，8ドットのうちのどのドットをセットし，どのドットをリセットにするかを決定し，2進数表現で1のビットに対応するドットはセット，0に対応するドットはリセットされます．〈タイルストリング〉がn文字の長さであれば，そのストリングの表わす模様は，このようにして決定される横8ドットのパターンを縦にn文字のパターン分だけ並べたものになります．従ってタイルストリングを構成する文字が全て同じ文字であったり，1文字だけであった場合には，縦線模様になります．

例）CHR$(&HAA)+CHR$(&H55)

16進数表現	2進数表現	ドットパターン
&HAA	10101010	●○●○●○●○
&H55	01010101	○●○●○●○●

上例は2文字からなる簡単なタイルストリングの例です．この2文字は互いに

1であるビットと0であるビットが全く逆になっています．このパターンを基本タイルとして指定された領域を埋めれば，細かい一松模様になりますが，PC-8801MKIIのドットが大変細かいために，その模様はほとんどわからず，灰色のように見えるでしょう．

カラーモードでも白黒モードと同じ様に，模様はタイルストリングに対応するビットパターンによって決定されますが，白黒モードと異なり3文字で横8ドットが構成されます．ストリング中の文字は先頭から青，赤，緑のプレーンのドットパターンを決定していきます．ストリングの長さに余りがあった場合には無視されますが，3文字に満たない場合には〝Illegal function call〟(FC Error) エラーとなります．

例）CHR$(&HAA)+CHR$(&H55)+CHR$(0)

上例は8×1ドットのパターンで決定するタイルストリングです．緑のプレーンのデータが0であるために，緑色は全く出ず，青と赤の縦線が交互に出る模様となります．しかし，ドットが細かいために縦縞はほとんどわからず，全体として紫色のようにみえます．このようにタイリングの利用により，基本8色を混ぜ合わせることで中間色の表現が可能です．

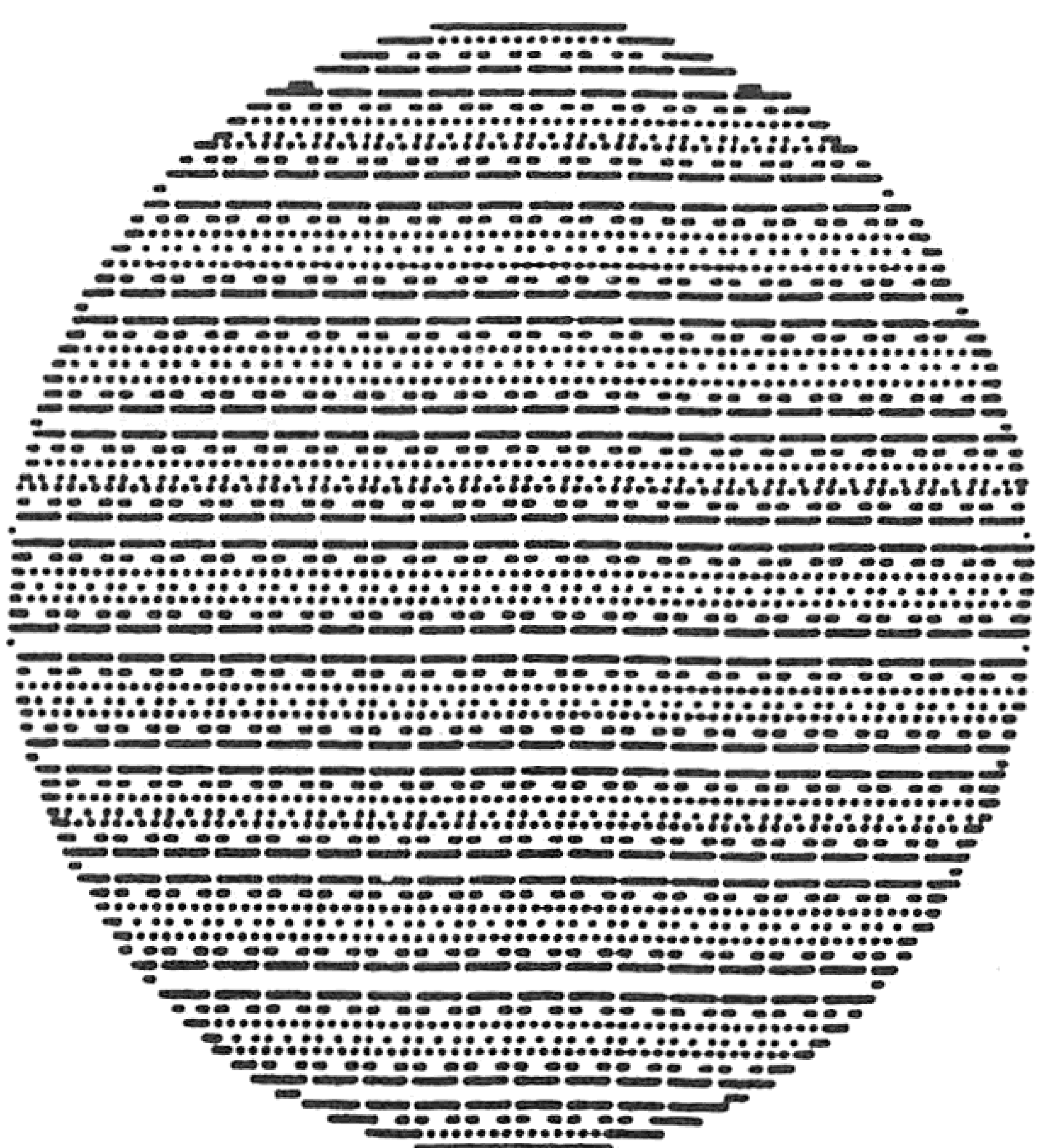
〈バックグラウンド•ストリング〉は，タイリングを行う領域に既に描かれている色，あるいはタイルパターンを指定します．この情報はタイリング実行時の補助的な情報で，バックグラウンド・ストリングが指定されていないとタイリング動作が中断されることがあります．省略された場合には，COLOR文によって指定されたバックグラウンドカラーに対応するストリングが用いられます．

〈バックグラウンド•ストリング〉は8ドット×1ドット分の長さ（白黒モードで1文字，カラーモードで3文字)が利用され，それを越える分は無視されます．また，〈バックグラウンド•ストリング〉が表す，8ドットのパターンが〈タイルストリング〉の表すパターン中に3回以上連続して現れる場合には，〝Illegal function call〟(FC Error) エラーとなります．

注意：既にタイリングされている領域を異なるパターンでタイリングしようとすると，時間がかかったり，スタックを消費するために〝Out of Memory〟(OM Error)エラーになったりすることがあります．タイリングが途中で止まったりする場合には，中断が生じた境界のパターンをバックグラウンド・ストリングとして指定したり，座標の指定を少し変えてやることにより中断を解消できます．

サンプルプログラム

```
100 '-- tiling paint (B/W mode) --
110 '
120 DATA 00,7f,63,55,49,55,63,7f
130 '
140 ' line 1 = 00000000 = &h00
150 ' line 2 = 01111111 = &h7f
160 ' line 3 = 01100011 = &h63
170 ' line 4 = 01010101 = &h55
180 ' line 5 = 01001001 = &h49
190 ' line 6 = 01010101 = &h55
200 ' line 7 = 01100011 = &h63
210 ' line 8 = 01111111 = &h7f
220 '
230 COLOR ,0:SCREEN 1,0,1,7:CLS 3
240 FOR I=1 TO 8
250  READ A$:TP=VAL("&h"+A$)
260  TILE$=TILE$+CHR$(TP)
270 NEXT I
280 CIRCLE(320,100),80,7
290 PAINT(320,100),TILE$,7
```



PEN ON/OFF/STOP　N88

機　能	ライトペンによる割り込みの許可，禁止，停止を設定する.
書　式	1)　PEN ON 2)　PEN OFF 3)　PEN STOP
文　例	PEN ON

解　説　ライトペンが押された時に起こる割り込みを許可 (ON) するか，禁止 (OFF) するか，停止 (STOP) するかを設定します.

1）割り込みを許可します．以後ライトペンを押すごとに割り込みを発生し，ON PEN GOSUB 文で定義されている処理ルーチンに分岐します.

2）割り込みを禁止します．以後ライトペンを押しても処理ルーチンへの分岐は起こりません.

3）割り込みを停止します．以後ライトペンを押しても，そのことを覚えているだけで処理ルーチンへの分岐は起こりません．しかしその後 PEN ON 命令によって割り込みが許可されると,先程のライトペンを押したことによって処理ルーチンに分岐します.

注意：プログラム終了時には PEN OFF を実行して割り込みを禁止しておいてください.

参照：ON PEN GOSUB，PEN

サンプル プログラム

```
PEN ON
PEN OFF
PEN STOP
```

POINT　N88

機　能	Last Referenced Point (LP)を変更する.
書　式	POINT { (Wx, Wy) / STEP(x, y) }
文　例	POINT(−200, 30)

解　説　N_{88}-BASICは，常に最後にグラフィック操作の行われた座標を覚えています．この座標は，相対座標により座標指定を行う際の基点となりますがPOINT文はこの座標を設定する働きをします．したがって次の例の1)と2)とは全く同じ意味を持つことになります．

1) LINE(−200, 30)−(100, 120), 3

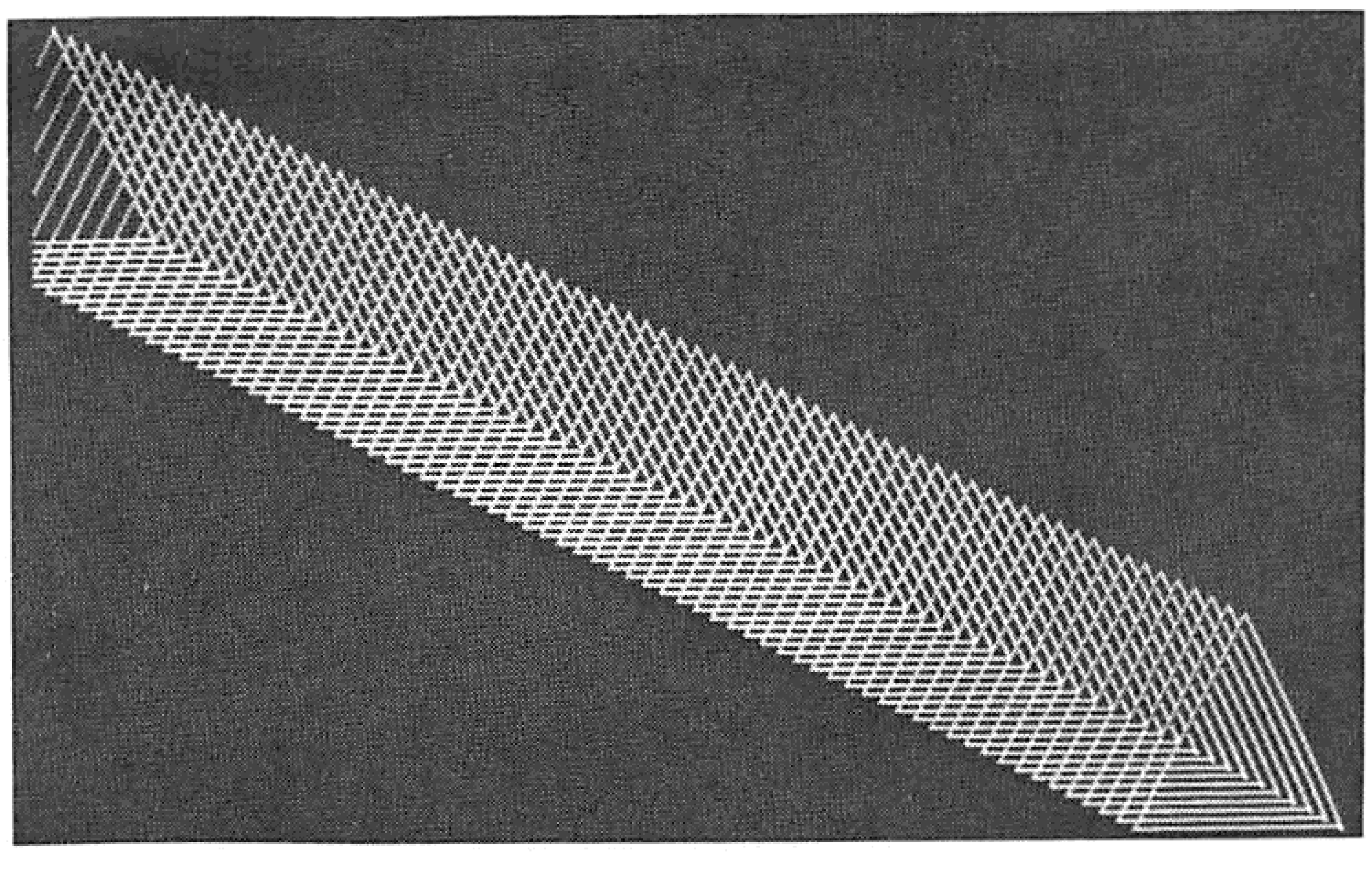
2) POINT(−200, 30)
　 LINE −STEP(300, 90), 3

STEPを付けると相対座標による指定となります（1章・座標の指定の仕方参照).

参照：LINE

サンプルプログラム

```
100 SCREEN 0,0:CLS 3
110 FOR I=0 TO 70
120  POINT STEP(8,2)
130  GOSUB *TRIANGLE
140 NEXT I:END
150 *TRIANGLE
160  LINE -STEP(-50,50),4
170  LINE -STEP(100,0),2
180  LINE -STEP(-50,-50),1:RETURN
```



POKE　N88　N

機　能	メモリ上の指定番地へデータを書き込む.
書　式	POKE 〈番地〉, 〈整数表記〉
文　例	POKE &HF000, &HFF

解　説　指定されたメインバンクのメモリ上の番地に，1バイト（8ビット）のデータを書き込みます.

〈整数表記〉は書き込まれるデータで，0～255（&H0～&HFF）の値でなければなりません．もし小数点以下があると整数化してから実行されます.

〈番地〉は2バイトの整数で，0～65535（&H0～&HFFFF）の値です.

但しこの命令は，現在のメモリの内容を書き換えてしまうため，不用意に使うとBASICが使っている作業領域を壊してしまい，誤動作の原因となることもあります．使う時には，メモリマップなどで使用可能な領域かどうかを確認してください.

POKE文の逆の働きをする関数としてPEEK関数というものも用意されています(関数のPEEK参照)．これらPOKEとPEEKは，効率的なデータの格納や機械語サブルーチンとの引数の受け渡しなどに役立ちます.

参照：PEEK

サンプルプログラム

```
100 CLEAR ,&HDFFF
110 '-- write for memory --
120 FOR ADD=&HE000 TO &HE00F
130  POKE ADD,DA:DA=DA+1
140 NEXT ADD
150 '-- read for memory --
160 FOR ADD=&HE000 TO &HE00F
170  PRINT RiGHT$(" 0"+HEX$(PEEK(ADD)),4);
180 NEXT ADD

run
 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Ok
```

PRESET　N88

機　能	画面上の任意のドットをリセットする.
書　式	PRESET | (Wx, Wy) | 〔, 〈パレット番号〉〕 | STEP(x, y) |
文　例	PRESET(100, 100)

解　説　ワールド座標の (Wx, Wy) で表されるドットをリセットします. STEP を付けた場合は相対座標による指定となります（1章・座標の指定の仕方参照).

この命令は, オプションの〈パレット番号〉を付けない方が一般的な使い方で, その場合, 現在 COLOR 文によって設定されているバックグラウンドカラーで点を塗り変えます.

〈パレット番号〉を指定した場合は, PSET 文と全く同じに機能し, そのパレット番号のカラーでドットをセットします. PRESET は LP を指定された座標 (Wx, Wy) に設定します.

参照：PSET, 〔1〕COLOR

サンプルプログラム

```
100 SCREEN 0,0:COLOR ,0:CLS 2
110 POINT(0,100)
120 FOR I=0 TO 200
130  PSET STEP(3,0),I MOD 8:NEXT I
140 POINT(0,100)
150 FOR I=0 TO 200
160  PRESET STEP(3,0)
170 NEXT I
```

PRESET　[N]

|機　能| 画面上のグラフィック座標（x，y）のドットをリセットします．
|書　式| PRESET（x，y〔，<ファンクションコード>〕）
|文　例| PRESET(100，50)

|解　説| PRESETは，<ファンクションコード>を指定しないのが，通常の使い方です．

注意：PRESETはグラフィックスイッチがONの状態で使用してください．

参照：PSET，〔1〕COLOR，CONSOLE

サンプルプログラム

```
100 CONSOLE ,,,1
110 PRINT CHR$(12)
120 FOR I=0 TO 90
130  PSET(I,I,C)
140  C=C+1
150  C=C MOD 8
160  PRESET(I,I)
170 NEXT I
```

PRINT／LPRINT　N88　N

機　能　ディスプレイまたはプリンタに情報を出力する.

書　式　1) PRINT〔〈式〉…〕

　　　　2) LPRINT〔〈式〉…〕

文　例　PRINT "ABC"

解　説　指定した式の値や文字列を1)はディプレイ画面に2)はプリンタに出力します．〈式〉が省略されている場合は改行のみを行います．式の値や文字列を表示する領域は，あらかじめ各行を14文字毎に分割して定められており，区切り記号にコンマを使うと次の領域の始めから，またセミコロンを使うと直前にプリントしたもののすぐ後ろに続いて次の値や文字列が表示されます．区切り記号として空白 "□" を用いた場合もセミコロンと同様の働きをします．

〈式〉の最後にセミコロン及びコンマを指定すると改行動作をおこさず，その行で次の PRINT 文による出力を続行します．

注意：変数と "で囲まれた文字列との区切りに限って ","，"；"，"□" を省略することができ，このときはセミコロンと同様の働きをします．

数値を表示した場合その後ろに必ず空白が一つ挿入されます．また，数の前に符号のための桁を確保します．単精度の数値で，指数形式でなくても 6 桁以下の桁数で精度に影響を及ぼさず表示できるものは，実数形式で表示されます．

同様に倍精度の数値は16桁以下の桁数で精度に影響を及ぼさず表示できるものは，実数形式の表示になります．

この命令のキーワード "PRINT" の簡略形として疑問符（？）を使用することが許されます．

表示する文字列の長さ，数値の長さが現在のカーソルの位置より後方にとれない場合（それを表示すると次の行にわたってしまう）改行してそれを表示します．

サンプルプログラム

```
100 FOR I=2 TO 15
110  FOR J=0 TO 15
120   PRINT CHR$(I*16+J);SPC(2);
130  NEXT J
140  PRINT
150 NEXT I
160 END

run
   !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /
0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?
@  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
P  Q  R  S  T  U  V  W  X  Y  Z  [  ¥  ]  ^  _
   a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
p  q  r  s  t  u  v  w  x  y  z  {  ¦  }  ~
▁  ▂  ▃  ▄  ▅  ▆  ▇  █  ▏  ▎  ▍  ▌  ▋  ▊  ▉  ┼
┴  ┬  ┤  ├  ▔  ─  │  ▕  ┌  ┐  └  ┘  ╭  ╮  ╰  ╯
   。  「  」  、  ・  ヲ  ァ  ィ  ゥ  ェ  ォ  ャ  ュ  ョ  ッ
ー  ア  イ  ウ  エ  オ  カ  キ  ク  ケ  コ  サ  シ  ス  セ  ソ
タ  チ  ツ  テ  ト  ナ  ニ  ヌ  ネ  ノ  ハ  ヒ  フ  ヘ  ホ  マ
ミ  ム  メ  モ  ヤ  ユ  ヨ  ラ  リ  ル  レ  ロ  ワ  ン  ゛  ゜
═  ╞  ╪  ╡  ◢  ◣  ◥  ◤  ♠  ♥  ♦  ♣  ●  ○  ╱  ╲
╳  円  年  月  日  時  分  秒
OK
```

PRINT #　N88 N

機　能	シーケンシャルファイルにデータを出力する.
書　式	PRINT # <ファイル番号>, 〔<式> …〕
文　例	PRINT #2, A, B, C

解　説　<ファイル番号>は，そのシーケンシャルファイルが出力モードとしてオープンされたときに指定された番号です.

但し，N-BASICでは，<ファイル番号>に－1を指定した場合には，カセットへの出力となります.

<式> はファイルに書き込まれる数値式または文字式です.

PRINT #文はデータの圧縮を行わず，PRINT文で画面へ出力するものと全く同じものを出力します．ですからこれらのデータがディスクから正しく入力できるように，ディスクに書き込まれたデータは適切に区切られるよう注意してください.

<式>のリストの中の数値式はセミコロンで区切るようにしてください．例えば，

```
PRINT #1, A ; B ; C ; X ; Y ; Z
```

(もし区切り記号にコンマを使うとプリント領域の間に挿入される余分な空白もディスクに書き込んでしまいます.)

リストの中の文字表記もセミコロンで区切ってください．ディスク上に文字表記を正しく書き込むためには，<式>のリストの中に独立した区切り記号を入れてください.

例をあげると，A$＝"CAMERA" そして，B$＝"93604－1" のときの次の文,

```
PRINT #1, A$ ; B$
```

はディスクにCAMERA93604－1と書き込みます．これは区切り記号がありませんから2つの別の文字列として入力することはできません．この問題を解決するには，次のようにPRINT #文中に独立に区切り記号を入れてください.

```
PRINT #1, A$ ; "," ; B$
```

これによってディスクに書き込まれるイメージは，

```
CAMERA, 93604－1
```

となり，2つの文字変数として読み込むことができます.

もし文字列それ自身がデータとしてコンマ，セミコロン，意味のあるはじめの空白，キャリッジリターン，またはラインフィードなどを含む場合は，別の引用符，CHR$(34)によって囲んでディスクに書いてください.

例をあげると, A$＝"CAMERA, □AUTOMATIC"また, B$＝"□□□93604－1" のとき，次の文,

PRINT＃1, A$；B$

は次のようなイメージをディスクに書きます.

CAMERA, □AUTOMATIC□□□93604－1

そして次の文,

INPUT ＃1, A$, B$

は, "CAMERA" をA$に，そして "AUTOMATIC 93604—1" をB$に読み込みます．これらの文字列をディスク上で正しく分割するには，CHR$(34) により引用符をディスクに書き込んで下さい．次の文,

PRINT ＃1, CHR$(34)；A$；CHR$(34)；CHR$(34)；B$；CHR$(34)

は次の内容をディスクに書き込みます.

"CAMERA, □AUTOMATIC" "□□□93604—1"

すると次の文,

INPUT ＃1, A$, B$

は "CAMERA, □AUTOMATIC" をA$に, "□□□93604—1" をB$に読み込みます.

PRINT #文は，USINGと共に使ってディスクファイルのフォーマットを制御することができます.

N_{88}- BASICでは, WRITE #文を使用すると適切にデータが区切られて書き込まれます.

参照：WRITE#, PRINT# USING, INPUT#

サンプルプログラム

```
100 '-- sequential data write --
110 '
120  OPEN "stest.dat" FOR OUTPUT AS #1
130  PRINT #1,DATE$;",";TIME$
140 *DATAWRITE
150   INPUT "ヒンメイ";NA$
160   IF NA$="end" THEN *EXIT1
170   INPUT "カカク";PLC
180   INPUT "コスウ";N%
190   PRINT #1,NA$;",";PLC;N%
200  GOTO *DATAWRITE
210 *EXIT1
220  PRINT:CLOSE
230 '-- sequential data read --
```

```
240  TOTAL=0
250  OPEN "stest.dat" FOR INPUT AS #1
260  INPUT #1,DA$,TI$
270  PRINT "ﾋﾂﾞｹ : ";DA$,"ｼﾞｶﾝ : ";TI$
280  PRINT
290 *DATAREAD
300   IF EOF(1) THEN *EXIT2
310   INPUT #1,NA$,PLC,N%
320   SUM=PLC*N%
330   PRINT NA$;TAB(10);PLC;"*";N%;"=";SUM
340   TOTAL=TOTAL+SUM
350  GOTO *DATAREAD
360 *EXIT2
370  PRINT:PRINT "TOTAL =";TOTAL
380 END

run
ﾋﾝﾒｲ? ｵｺﾒ
ｶｶｸ? 2500
ｺｽｳ? 1
ﾋﾝﾒｲ? ﾗｰﾒﾝ
ｶｶｸ? 65
ｺｽｳ? 5
ﾋﾝﾒｲ? ｷﾞｭｰﾆｸ
ｶｶｸ? 800
ｺｽｳ? 2
ﾋﾝﾒｲ? end

ﾋﾂﾞｹ : 81/01/01              ｼﾞｶﾝ : 19:59:30

ｵｺﾒ       2500 * 1 = 2500
ﾗｰﾒﾝ       65 * 5 = 325
ｷﾞｭｰﾆｸ     800 * 2 = 1600

TOTAL = 4425
Ok
```

PRINT% [N]

機　能	RS-232C通信ポートの入出力バッファにデータを出力します．
書　式	PRINT%〈ポート番号〉，〔〈式〉……〕
文　例	PRINT%1，A，B

解　説　〈ポート番号〉で指定されたRS-232C通信ポートの入出力バッファにデータを出力します。使い方はフロッピィディスクへ出力するPRINT＃と同じです．

〈ポート番号〉は本体内蔵のRS-232Cインタフェースを使用する場合に1を指定します．

参照：INPUT％，INIT％

サンプルプログラム

```
100 '-- RS-232C port output --
110 'switch to RS-232C channel
120 OUT &H30,&H20 OR (&HF AND PEEK(&HEA66))
130 POKE &H8001,&H23:POKE &H8000,&H7F
140 OUT &HE6,4
150 OUT &HE4,&HFF
160 INIT %1,&HFA,&H37
170 LINE INPUT "message?";A$
180 IF A$="end" THEN OUT &HE6,0:END
190 PRINT %1,A$
200 GOTO 170
```

PRINT USING／LPRINT USING　N88　N

機　能	文字列，数値を指定した書式で出力する.
書　式	PRINT / LPRINT　USING 〈書式制御文字列〉；〈式〉〔 ; / , 〈式〉…〕〔 ; / , 〕
文　例	PRINT　USING　"#####,."；A，B

解　説　〈書式制御文字列〉によって後に続く〈式〉の出力される領域や書式を決定します.

但し，"_" と "@" の2つの機能はN-BASICにはありません.

文字変数や文字列の書式制御

- ！……与えられた文字変数や文字定数の最初の1文字だけ出力します.
- ＆〈n個の□（空白）〉＆……与えられた文字変数や文字定数の先頭から（n＋2）文字の文字列を出力します. 与えられた文字列が（n＋2）文字よりも長い場合は，余分な文字は無視され，短い場合には，文字列は左づめに出力され，残った部分には空白が出力されます.
- @……文字列の代入に使います. 複数個指定した場合，1つの "@" に対して〈式〉中の1つの文字列が，代入・出力されます. "@" の数が文字列の個数より多い場合，残りの "@" は無視されます.

数値の書式制御

- #……数値を出力する桁数を指定します. 指定した桁数より数値の桁数が小さいときには，右づめで出力されます.
- .……小数点の位置を指定します. 小数点以下の部分で冗長となる桁には0が出力されます.
- ＋……〈書式制御文字列〉の最初または最後に付けた場合，数値の符号がそれぞれ前または後に出力されます. 2個以上の "＋" を並べた場合には，余分は後述の制御文字以外の文字と同じ扱いになります.
- －……〈書式制御文字列〉の最後に付けた場合，数値が負の数の時に数値の後ろに "－" が出力されます. 前に付けたり，2個以上並べた場合には後述の制御文字以外の文字と同じ扱いになります.
- ＊＊……〈書式制御文字列〉の先頭につけた場合，数値領域の左側に空白部分ができたとき，そこを "＊" で埋めて出力します. この "＊＊" は2桁分の領域を確保します.

● ¥¥……〈書式制御文字列〉の先頭につけた場合，出力される数値の直前に〝¥〞を出力します．〝¥¥〞は2桁分の領域を確保しますが，このうち1桁分は〝¥〞の出力領域として使われます．後述の指数形式の書式指定を行った場合には，正しく出力されない事があります．

● ＊＊¥……〈書式制御文字列〉の先頭につけた場合，上記の2つ（＊＊と¥¥）の両方の機能となります．〝＊＊¥〞は3桁分の領域を確保しますが，このうち1桁分は〝¥〞の出力領域として使われます．

● ，……桁数指定の〝#〞の並びの中においた場合，数値の整数部が3桁毎に〝,〞で区切られて出力されます．但し，上記の〝.〞より右側においた場合は，数値の最後に〝,〞が出力され，3桁毎の区切りは行われません．

● ∧∧∧∧……桁数指定の〝#〞の後に付けた場合，数値は指数形式で出力されます．

● _……上記の制御文字1文字を単に文字として表示するために使用します．〝_〞に続く1文字は常に書式制御機能を持たない文字として出力されます．

制御文字以外の文字

以上の制御文字以外の文字（英数字，カナ，グラフィック記号等）を置いた場合，数値の前や後ろにそのキャラクタが出力されます．

数値の領域をこえた場合

指定した数値領域より数値の桁数が大きい場合，数値の直前に〝%〞が出力されます．数値の丸めが領域より大きくなる原因となった場合も，丸めた数値の前に〝%〞が出力されます．

サンプルプログラム

```
100 PRINT USING "!";"NEC PC-8801MK2"
110 PRINT USING "&          &";"NEC PC-8801MK2"
120 PRINT USING "#####";123.456
130 PRINT USING "#####.#";123.456
140 PRINT USING "+####.#";123.456
150 PRINT USING "#####.#-";-123.456
160 PRINT USING "**###.#";123.456
170 PRINT USING "¥¥###.#";123.456
180 PRINT USING "**¥##.#";123.456
190 PRINT USING "#####,.#";1234.56
200 PRINT USING "#####,.#^^^^";1234.56
210 PRINT USING "¥¥###._-";123.456
220 PRINT USING "#####";123456!
230 PRINT USING "This is a @.";"pen"
240 END
```

```
run
N
NEC PC-88
   123
   123.5
  +123.5
   123.5-
**123.5
 ¥123.5
*¥123.5
 1,234.6
 12345.6E-01
 ¥123.-
%123456
This is a pen.
Ok
```

PRINT # USING　N88 N （Nではディスクのみ）

機能	文字列，数値を指定した書式でファイルに出力する．
書式	PRINT # 〈ファイル番号〉，USING 〈書式制御文字列〉；〈式〉 [; , 〈式〉…] [; ,]
文例	PRINT #2, USING "####"；A

解説　〈ファイル番号〉で指定されたファイルに対して，文字列や数値を書式指定して出力します．

PRINT # USINGは，その対象がファイルであることを除けばPRINT USINGと全く同じです．

参照：PRINT USING，PRINT #，OPEN

サンプルプログラム

```
100 OPEN "stest.dat" FOR OUTPUT AS 1
110 FOR I=0 TO 10
120   PRINT #1,USING "####";I^2
130 NEXT I
140 END
```

PSET　N88

機　能	画面上の任意のドットをセットする.
書　式	PSET | (Wx, Wy) / STEP(x, y) | 〔, 〈パレット番号〉〕
文　例	PSET(100, 100), 4

解　説　ワールド座標の(Wx,Wy)の位置にドットをセットします. STEPを付けると相対座標による指定となります（1章・座標の指定の仕方参照).

カラーは〈パレット番号〉によって指定し，もし省略された場合は，現在のフォアグラウンドカラー（COLOR文参照）が採用されます.

PSETはLPを指定された座標（Wx, Wy)に設定します.

参照：〔1〕COLOR, PRESET

サンプルプログラム

```
100 SCREEN 0,0:CLS 2
110 COLOR ,,,4
120 PSET(100,100)
130 '
140 POINT(150,100)
150 FOR I=0 TO 7
160  PSET STEP(10,0),I
170 NEXT I
```

PSET [N]

機能	画面上の任意のドットをセットする.
書式	PSET(x, y〔,ファンクションコード〕)
文例	PSET(100, 50, 7)

解説　画面上のグラフィック座標（x, y）にドットをセットします.

オプションの〈ファンクションコード〉はCOLOR(N-BASIC)文に用いる場合と同じで，カラーモードの時は色を指定することができ，白黒モードの時はリバースなどの機能を指定することができます.

注意：PSETはグラフィックスイッチがONになっている状態で使用してください.

参照：PRESET，COLOR[N]，CONSOLE

サンプルプログラム

```
100 CONSOLE ,,,1
110 PRINT CHR$(12)
120 FOR I=0 TO 90
130  PSET(I,I,C)
140  C=C+1
150  C=C MOD 8
160 NEXT I
```

PUT　N88 N　（Nではディスクのみ）

機能　バッファ中のデータをファイルに書き出す.

書式　PUT〔#〕〈ファイル番号〉〔,〈数〉〕

文例　PUT #3, 5

解説　〈ファイル番号〉で指定されたファイルに対応するバッファの内容を書き出します. PUT は, 指定されたファイルがディスクファイルであるかどうかで, その動作が異なります.

〈ファイル番号〉で指定されたファイルがディスクファイルの場合には, PUT は, ランダムファイルへの書き込みとして働きます. 指定されたファイルは, ランダムファイルのモードでオープンされていなければなりません. 〈数〉はファイルのレコード番号として解釈され, バッファ中のデータ256バイトが指定されたレコードに書き込まれます. レコード番号が省略された場合には, 直前に行われた, GET, PUT で参照されたレコードの次のレコードに書き込まれます.

〈ファイル番号〉で指定されたファイルがディスクファイルでない場合には, PUT は, バッファ中のデータを指定されたファイルに対して出力する働きをします. この時, 指定されたファイルは出力モードでオープンされていなければなりません. 〈数〉はバッファから, ファイルに対して書き出す文字数を意味し, 0から255までの値で指定します. 〈数〉が省略された時, 及び0が指定された時には, 256文字を書き出します.

なお書き出すデータは予め FIELD, LSET／RSET により準備しておかねばなりません.

N-BASIC では, PUT はディスクのランダムアクセス・ファイルのみに用いることができます.

参照：OPEN, FIELD, GET, LSET／RSET

サンプル
プログラム

```
100 '-- random data write --
110 OPEN "1:data1" AS #1
120 FIELD #1,30 AS A$,20 AS B$,50 AS C$
130 *ENTRY
140 LINE INPUT "NAME?";NAM$
150 IF NAM$="end" THEN *EXIT
160 LINE INPUT "TEL?";TEL$
170 LINE INPUT "ADDRESS?";ADR$
180 LSET A$=NAM$
190 LSET B$=TEL$
200 LSET C$=ADR$
210 PUT #1
220 GOTO *ENTRY
230 *EXIT
240 CLOSE

run
NAME?ｷﾉｼﾀ
TEL?00-045-6753
ADDRESS?ｵｵｻｶﾌ ｵｵｻｶｼ 1-49-5
NAME?ｱｵｷ
TEL?00-065-8734
ADDRESS?ﾄｳｷｮｳﾄ ｼﾝｼﾞｭｸｸ 3-8-6
NAME?ｵｵﾀ
TEL?00-087-5619
ADDRESS?ﾄｳｷｮｳﾄ ｼﾌﾞﾔｸ 3-10-7
NAME?end
Ok
```

PUT@ N88

機　能　グラフィックパターンや漢字を画面に表示する.

書　式　1) PUT〔@〕(Sx, Sy),〈配列変数名〉〔〈要素ナンバー〉〕〔,〈条件〉〕
〔,〈フォアグラウンドカラー〉,〈バックグラウンドカラー〉〕
2) PUT〔@〕(Sx, Sy), KANJI (〈漢字コード〉),〔,〈条件〉〕
〔,〈フォアグラウンドカラー〉,〈バックグラウンドカラー〉〕

文　例　1) PUT(100, 100), G%, PSET
2) PUT(0, 0), KANJI(12321)

解　説

1）GET@命令によって配列に読み込まれたグラフィックパターンを画面上の任意の位置に表示します.

座標(Sx, Sy)はGET@の場合と同様で，ワールド座標ではなくスクリーン座標で指定します．またこの命令を実行するとLPは(Sx, Sy)に移動されます.

〈配列変数名〉は表示したいグラフィックパターンが格納されている配列名であり，〈要素ナンバー〉は配列内のどこからデータを取り始めるかの指定です．〈要素ナンバー〉が省略された場合は，配列の最初から取り始めます．これらの指定は，GET@命令で使ったものと同じものを指定します.

〈条件〉とは，グラフィックパターンを画面に表示する際，いろいろな機能を指定できるもので，次下のものが用意されています.

PSET　　配列内のグラフィックパターンをそのまま表示します.

PRESET　　白黒モードの場合は配列内のパターンをリバースして表示します．カラーモードの場合は各ドットのパレット番号を，7－(そのドットのパレット番号）として表示します.

OR　　配列内のグラフィックパターンと，既にある画面上のグラフィックパターンを1ビット（ドット）ごとにOR（論理和）し，その結果を画面に表示します.

AND　　配列内のパターンと画面上のパターンをビットごとにAND（論理積）し，その結果を画面上に表示します.

XOR　　配列内のパターンと画面上のパターンをビットごとにXOR（排他的論理和）し，その結果を画面に表示します.

※これらの〈条件〉は画面モードによって演算の対象が違います．白黒モードの場合，ドットがあるかないか（1ビット）を対象とし，カラーモードの場

合各パレット番号（0〜7：3ビット）を対象とします.

例）（パレット3）AND（パレット6）→（パレット2）

もし〈条件〉が省略された場合はXORとみなされます.

最後の〈フォアグラウンドカラー〉,〈バックグラウンドカラー〉は，白黒モードの時に読み込んだパターンをカラーモードにおいて表示する時にのみ有効なオプションパラメータです．この2つのパラメータは両方とも指定するか，両方とも指定しないかのどちらかしか許されません.

〈フォアグラウンドカラー〉は，白黒モードで読み込んだ際に白であったドットに対しての色指定で，これをカラーモードにおいて実行すると任意の色に変えることができます．〈バックグラウンドカラー〉は同様に黒であったドットに対しての色指定です．これらは共に0から7のパレッド番号によって指定します.

注意：GET@とPUT@命令は上記した〈フォアグラウンドカラー〉,〈バックグラウンドカラー〉を指定して白黒モードで読み込んだパターンをカラーモードで表示する用途の他は,原則として同一画面モードで使用するようにしてください.

2）〈漢字コード〉で指定された漢字または非漢字を画面に表示します.

使用できる文字は，JIS第1水準の漢字2965文字と非漢字700文字です．ユーザーはこれらの中から任意の文字を〈漢字コード〉(ユーザーズ・マニュアル参照）によって指定し，画面上に日本語の文章を作成することができます.

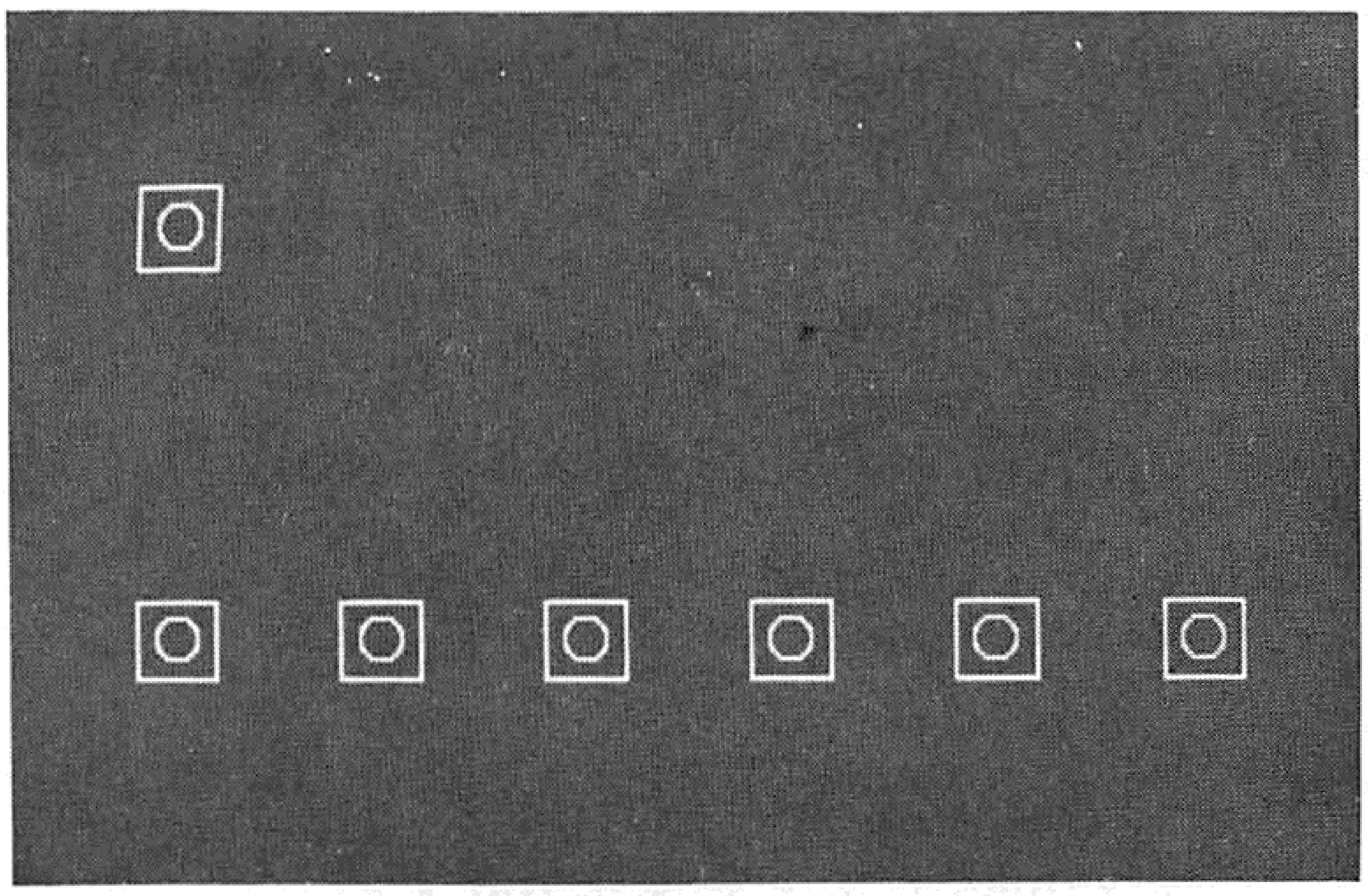
1)のPUT文ではユーザーがGET文によって配列に読み込んだパターンを画面に表示しますが,漢字のPUT文では配列内のデータにあたるものが,あらかじめ用意されています．このことを除けば，その使い方及び機能は1)の場合とほぼ同様です．但し，漢字パターンは原則として縦16ドット，横16ドットの大きさと決められています（8×8，16×8のものもある。ユーザース・マニュアル参照)．また漢字パターンは白黒モードでGETした場合と同じ形で格納されていますから，カラーモードにおいて〈フォアグラウンドカラー〉,〈バックグラウンドカラー〉で指定してカラー表示させることも可能です.

注意：1つのPUT文字では1つの漢字しか表示することができません.

参照：GET@

サンプルプログラム

```
100 SCREEN 0,0:CLS 3
110 XD=40:YD=20
120 BYTE=((XD+7)¥8)*YD*3+4
130 FACT=BYTE¥2+1
140 DIM G%(FACT)
150 '
160 LINE(0,0)-STEP(XD-1,YD-1),1,B
170 CIRCLE(XD/2-1,YD/2-1),YD/2,4
180 GET(0,0)-STEP(XD-1,YD-1),G%
190 '
200 FOR X=0 TO 500 STEP 100
210 PUT(X,100),G%:NEXT
```



```
100 '-- kanji print --
110 DATA 427e,3a23,3441,3b7a,244e
120 DATA 2546,2539,2548,4366,212a
130 '
140 SCREEN 0,0:CLS 3
150 X=150
160 FOR I=1 TO 10
170  READ A$:KCODE=VAL("&h"+A$)
180  PUT(X,80),KANJI(KCODE),PSET,7,0
190  X=X+35
200 NEXT I
```

只　今　漢　字　の　テ　ス　ト　中　！

PUT@　Ⓝ

機　能　GET@文で配列へ読みこんだキャラクタやドットグラフィックスを画面の任意の位置へ表示させます.

書　式　1)　PUT@(X_1, Y_1)－(X_2, Y_2), 〈配列名〉, 〈条件〉

2)　PUT@A (X_1, Y_1)－(X_2, Y_2), 〈配列名〉

文　例　PUT@(50, 50)－(100, 80), A%

解　説

1) GET@文でキャラクタとして配列へセーブしたものを表示させる場合水平位置X_1, X_2は0から（1行の桁数)－1まで, 垂直位置Y_1, Y_2は0から（画面の行数)－1までの値をとります. そして〈条件〉には色やブリンクなどを指定するための〈ファンクションコード〉を置くことができます.

GET@文でドットグラフィックスとして配列へセーブしたものを表示させる場合のX_1, X_2は0から（1行の桁数)＊2－1までの値, Y_1, Y_2は0から（画面の行数)＊4－1までの値をとります. そして〈条件〉にはPSET, PRESET, OR, AND, NOT, XORのいずれかを指定しなければなりません.

GET@Aでセーブした配列をAの付かないこの書式で表示させることはできません.

2) GET@文の2)の方法でセーブしたものを表示させる.

水平位置X_1, X_2は0から(1行の桁数)－1までの値, 垂直位置Y_1, Y_2は0から(画面の行数)－1までの値をとります. 色などもセーブした時点のものを自動的に与えます.

GET@文の1)でセーブしたものをこの書式で表示することはできません.

参照：Ⓝ GET@

サンプルプログラム

```
100 DIM G%(100)
110 LINE(0,0)-(10,10),PSET,BF
120 GET@(0,0)-(10,10),G%,G
130 PUT@(100,50)-(110,60),G%,PSET
```

RANDOMIZE　N88

機　能	新しい乱数系列を設定する.
書　式	RANDOMIZE〔<式>〕
文　例	RANDOMIZE

解　説　新しい乱数の種（seed）を与えることによって乱数系列を変更します. 乱数の seed は〈式〉によって－32768から32767の間で与えます. もし省略された場合, メッセージが出力されて, seed を要求してきます.

この場合も〈式〉と同じ範囲の値を入力しなければなりません. もしこの範囲を越えると〝Overflow″（OV Error）エラーとなります.

参照：RND

サンプルプログラム

```
100 GOSUB *RNDSUB
110 CLEAR:GOSUB *RNDSUB
120 RANDOMIZE
130 GOSUB *RNDSUB
140 END
150 *RNDSUB'-- rnd number print
160  FOR I=0 TO 9
170   PRINT INT(RND*10);
180  NEXT I:PRINT
190 RETURN

run
 2  3  3  5  0  7  4  3  9  9
 2  3  3  5  0  7  4  3  9  9
Random number seed (-32768 to 32767)? 653
 0  2  4  9  6  5  2  7  6  7
Ok
```

R

READ　N88　N

機　能	DATA 文より値を読み，変数に割り当てる.
書　式	READ <変数リスト>
文　例	READ K, M, S$

解　説　READ 文はいつでも DATA 文と組み合わせて使わなければなりません. READ 文は DATA 文のデータを 1 対 1 対応の方法で変数に割りあてていきます. READ 文の変数は数値変数でも文字変数でもかまいません. しかし読み出された値と変数の型は一致していなければなりません. もし，型が一致しない場合には，〝Syntax error〟(SN Error) が起こります.

1 つの READ 文が 1 つまたはそれ以上の DATA 文を参照したり (複数の場合は順番に参照されます)，またいくつかの READ 文が 1 つの DATA 文を参照することができます. もし<変数のリスト>の変数の数が DATA 文のデータの個数を越えてしまった場合は，〝Out of DATA〟(OD Error) のエラーメッセージが表示されます. 指定された変数が DATA 文のデータの個数よりも少ない場合には，その次の READ 文が読まれなかったデータから読み始めます. もしそれ以上 READ 文がない場合には，余分のデータは無視されます.

始めから DATA 文を読みなおすには，RESTORE 文を使います.

参照 : RESTORE, DATA

サンプルプログラム

```
100 FOR I=0 TO 10
110  READ A
120  PRINT A;
130 NEXT I
140 READ A$:PRINT A$
150 END
160 DATA 10,9,8,7,6,5,4,3,2,1,0,Fire !

run
 10  9  8  7  6  5  4  3  2  1  0 Fire !
Ok
```

REM　N88 N

機能　プログラムに注釈を入れる.

書式　REM〔<注釈文>〕

文例　REM TESTPROGRAM
' TESTPROGAM

解説　REM文は非実行文であり，プログラムの実行に影響を与えずコメント行とすることができます．リストを取ると入力した内容がそのまま出力されます．但し，REM文はGOTO文，GOSUB文の飛先として使うことは可能です．

REM文ではキーワード〝REM〟の代わりにアポストロフィ（'）を使うことができます．REM文の行はコロンで区切って他の文を続けることはできません．

サンプルプログラム

```
100 '   REM ｽﾃｰﾄﾒﾝﾄ ﾆ ﾖｯﾃ
110 ' ﾌﾟﾛｸﾞﾗﾑ ﾉ ﾅｶﾆ
120 ' ｺﾒﾝﾄ ｶﾞ ｶｸﾏｽ.
130 '
140 '   REM ﾊ ｺﾉ ﾚｲ ﾉ ﾖｳﾆ
150 ' "'" ﾃﾞ ﾀﾞｲﾖｳ ﾃﾞｷﾏｽ.
160 END
```

REMOVE（ディスク） N

機　能	ディスクにファイル配置表を書き込んでディスク入出力を終了する.
書　式	REMOVE〔<ドライブ番号>〔, <ドライブ番号> …〕〕
文　例	REMOVE 1

解　説　REMOVE は MOUNT 命令の逆の動作をする命令で，ディスケットをドライブから取り出す前に必ず実行しなければなりません.

REMOVE 命令を実行すると現在メモリ上にある配置表を，コピーと合わせて3組，ディスケットのディレクトリトラックに書き込み，MOUNT 命令と同じように誤りを検出します.

注意：ディスク入出力を終了し，ドライブからディスケットを取り出す時には，必ず REMOVE 命令を実行してください．もし怠ると，取り出したディスケットの配置表は更新と誤り検出が行われず，またそのドライブに入れられた次のディスケットには誤った配置表が書き込まれてしまい，両方とも破壊されてしまいます.

また，REMOVEは，最初の3文字が同じならば〝OK〞を出力します（REM文とみなされます）が，実際には〝REMOVE〞の動作は行なわれていません.
（REMOVEを実行するときは，つづりをまちがえないように注意してください.

例） REM, REMO, REMVEなど……REMOVEとしてみなされない.

N_{88}- BASIC ではオートマチック MOUNT, REMOVE を採用しているため，これらの操作はすべて BASIC が管理します.

参照：MOUNT

サンプルプログラム

```
remove 1
remove
```

RENUM　N88 N

機能	プログラムの行番号を整理する.
書式	RENUM〔<新行番号>〕〔, <旧行番号>〕〔, <増分>〕
文例	RENUM

解説　<新行番号>は，新しくつける行番号の最初の行番号で，省略値は10です.<旧行番号>は行番号のつけ替えをはじめる現在のプログラムの行番号です.省略値はそのプログラムの最初の行番号です. <増分>は新しく付ける各行番号のあいだの増分で，省略値は10です.

RENUM コマンドは，また，GOTO, GOSUB, THEN, ON…GOTO, ON…GOSUB 及び ERL 文などで参照している行番号も新しい行番号に対応して変更します. これらの文が参照している行番号の行がもし存在しない場合には，"Undefined line xxxxx in yyyyy" (UN Error) とエラーメッセージが表示されます. この場合，誤った行番号(xxxxx)は RENUM コマンドによって変更されませんが，行番号 yyyyy は変更されます.

注意：RENUM コマンドをプログラム行の順序を変えるのに使うことはできません (たとえば，10, 20, 30の 3 つの行がある場合の RENUM 15, 30 など). また65529以上の行番号を発生することもできません. このような場合には "Illegal function call" (FC Error) エラーが起こります.

サンプルプログラム

```
100 ' RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｶﾗ
110 ' ｷﾞｮｳﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴﾏｽ.
120 '
130 PRINT " RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｶﾗ"
140 PRINT " ｷﾞｮｳﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴﾏｽ."
150 END
Ok
renum 200,130,20
Ok
list
100 ' RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｶﾗ
110 ' ｷﾞｮｳﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴﾏｽ.
120 '
200 PRINT " RENUM ｺﾏﾝﾄﾞ ﾊ ｼﾃｲ ｼﾀ ｷﾞｮｳ ｶﾗ"
220 PRINT " ｷﾞｮｳﾊﾞﾝｺﾞｳ ｦ ﾂｹｶｴﾏｽ."
240 END
Ok
```

RESTORE　N88　N

機　能　READ 文で読む DATA 文を指定する.

書　式　RESTORE〔〈行番号〉〕

文　例　RESTORE 800

解　説　〈行番号〉が省略されると，次にくる READ 文はプログラム中の最初の DATA 文から読み始めます. 〈行番号〉を指定すると，指定された行の DATA 文から読み始めます.

サンプルプログラム

```
100  READ A,B,C
110 RESTORE
120  READ D,E,F
130 RESTORE *STRDATA
140  READ A$,B$,C$
150  PRINT A,B,C
160  PRINT D,E,F
170  PRINT A$,B$,C$
180 END
190  DATA 1,2,3
200  DATA 4,5,6
210 *STRDATA
220  DATA aa,bb,cc

run
 1              2              3
 1              2              3
aa             bb             cc
Ok
```

RESUME　N88 N

機　能	エラー回復処理終了後，プログラムの実行を再開する．
書　式	RESUME ［0］ / NEXT / 〈行番号〉
文　例	RESUME 0 RESUME NEXT RESUME 1000

解　説　プログラムの実行を再開する場所に応じて次のように3つの書式を選ぶことができます．

RESUME または RESUME 0	エラーの原因となった文からプログラムの実行が再開されます．
RESUME NEXT	エラーの原因となった文のすぐ次の文から実行が再開されます．
RESUME 〈行番号〉	〈行番号〉で指定した行から実行が再開されます．

参照：ON ERROR GOTO

サンプルプログラム

```
100  ON ERROR GOTO *ERRSUB
110 *ENTRY
120  INPUT "X,Y";X,Y
130  A=X/Y
140  PRINT "X/Y =";A
150  GOTO *ENTRY
160 *ERRSUB
170  PRINT "** ｹｲｻﾝ ﾃﾞｷﾏｾﾝ **
180  RESUME *ENTRY

run
X,Y? 100,10
X/Y = 10
X,Y? 35,0
** ｹｲｻﾝ ﾃﾞｷﾏｾﾝ **
X,Y? 35,5
X/Y = 7
X,Y?
Break in 120
Ok
```

ROLL（ディスク） N88

機　能	**グラフィック画面をスクロールアップさせる.**
書　式	ROLL 〈ドット数〉
文　例	ROLL 16

解　説　グラフィック画面を指定された〈ドット数〉だけ上にスクロールします.

〈ドット数〉とは画面の縦のドットの数です. 許される値は1から197までの正の値で, 負の値を与えてスクロールダウンさせることはできません.

この命令は漢字など, 日本語の文章を表示するときのスクロールに使うことができます.

参照 : PUT @

サンプルプログラム

```
100 SCREEN 1,0:CLS 3
110 FOR I=&H3000 TO &H5000 STEP &H100
120  FOR J=&H21 TO &H7E
130   KCODE=I+J
140   PUT(X,168),KANJI(KCODE),PSET
150   X=X+20
160   IF X>623 THEN X=0:ROLL 18
170  NEXT J
180 NEXT I
```

RUN　N88　N

機　能　メモリにあるプログラムの実行を開始する.
ディスクからファイルをメモリにロードし，そのプログラムを実行する.

書　式　1）RUN〔〈行番号〉〕
2）RUN〈ファイルディスクリプタ〉〔，R〕

文　例　1）RUN 100
2）RUN "2：TEST"

解　説

1）〈行番号〉を指定すると，その行から実行がはじまります．指定のない場合には，最も若い行番号の行から実行がはじまります．プログラムの実行が終るとBASICはいつでもコマンドに戻ります．

2）〈ファイルディスクリプタ〉で指定されるプログラムをロード後実行を開始します．

RUNコマンドを実行すると，すべての開いているファイルは閉じ，目的のプログラムをロードする前にメモリの内容をクリアします．しかし，Rオプションを付けた場合には，すべてのデータファイルは開いたままになります．

サンプルプログラム

```
RUN
RUN 1000
RUN *START
RUN "SAMPLE"
RUN "2:TEST",R
```

SAVE　N88 N（Nではディスクのみ）

機　能	メモリのBASICプログラムをファイルにセーブする.
書　式	SAVE <ファイルディスクリプタ>〔, {A / P}〕
文　例	SAVE "2 : MYPROG", A

解　説　<ファイルディスクリプタ>で指定されるファイルにプログラムを書き込みます. 指定したファイル名と同じ名前のファイルが存在した場合には, 古い内容は失なわれて, 新しいものに更新されます.

Aオプションが指定された場合には, プログラムはアスキー型式でセーブされます. オプションの指定がない場合には, バイナリ型式に圧縮されてプログラムのセーブが行われます. アスキー型式のセーブは, バイナリ型式よりも, 多くのファイルスペースを必要としますが, セーブされたプログラムファイルを操作する場合には, アスキー型式でセーブされていることを必要とします. 例えばMERGEコマンドは, アスキー型式のファイルを必要とします. また, アスキー型式でセーブされたファイルは, データファイルとして読み出すことができます.

Pオプションを指定した場合には, プログラムは暗号化されたバイナリ型式でセーブされます. Pオプションは, セーブされたプログラムを, 書き込み, 変更といった操作から保護するための機能を持っています. また後で, メモリにロードしてもLISTやEDITにより, 内容を見たり, 変更しようとすると, "Illegal function call" (FC Error) エラーとなります. 一度, Pオプションにより保護が指定されると, これを解除する方法は用意されていませんので, 二度と内容の変更等はできなくなるので注意してください.

N-BASICでは, SAVEコマンドは, ディスク版でのみ用意されており, ディスクファイルに対してのみセーブすることができます. またN-BASICではPオプションの機能は含まれていません.

参照：LOAD, MERGE

S

サンプルプログラム

```
save "cas:sample1"
save "1:sample3"
save "2:sample4",p
```

SCREEN　N88

機　能　**グラフィック画面に対して種々のモードを設定する.**

書　式　SCREEN 〔〈画面モード〉〕〔,〈画面スイッチ〉〕〔,〈アクティブページ〉〕〔,〈ディスプレイページ〉〕

文　例　SCREEN 1, 0, 0, 7

解　説　カラーモードにするか白黒モードにするか，白黒モードでどのページを書き込み及び表示ページとするかなど，ディスプレイのグラフィック画面に対しての種々のモードを設定します.

〈画面モード〉は，カラー，白黒，分解能など，グラフィック画面の最も基本的なモードを設定するパラメータで次の値をとります.

0――カラーモード（640×200ドット）
1――白黒モード　（640×200ドット，3ページ）
2――高分解能白黒モード（640×400ドット）

〈画面スイッチ〉は，1ビットで表現される，〝グラフィックマスクスイッチ〟と〝高速書き込みスイッチ〟の2つのスイッチを合わせ持ったパラメータです.

〝グラフィックマスクスイッチ〟をONにすると，現在グラフィック画面に描かれているパターンなどは一時的に消去されます. OFFにすれば元に戻ります.

〝高速書き込みスイッチ〟は，CPUがグラフィックRAMに書き込む時のモードを設定するもので，ONにすると通常より高速で書き込みます. 但し，ONの状態でグラフィック命令を実行すると画面がチラツキます. よほど高速性が要求される場合以外は，グラフィックマスクスイッチをOFFにした状態で使用するようにしてください.

この2つのパラメータは次のように2ビットのビット対応になっており，与える値としては次のように0から3までとなります.

ビット1　ビット0
グラフィックマスクスイッチ→□□←**高速書き込みスイッチ**

0――→[0|0]　高速書き込み，グラフィックマスクスイッチ共OFF.
1――→[0|1]　高速書き込みスイッチのみON.
2――→[1|0]　グラフィックマスクスイッチのみON.
3――→[1|1]　高速書き込み，グラフィックマスクスイッチ共ON.

〈アクティブページ〉と〈ディスプレイページ〉は白黒3ページモードの際の書き込むページと表示ページの選択であり，カラーモードと高分解能白黒モードでは意味を持ちません．

〈アクティブページ〉は今後グラフィック命令によって書き込むページを0から2で指定します．〈ディスプレイページ〉は3ページのうちどのページを表示するかを指定します．この指定は各ページがビットに対応した3ビットの2進数表現（0～7）で行います．

ビット2	ビット1	ビット0
ページ3	ページ2	ページ1

0 ──→ [0|0|0]　全ページ表示しない．

1 ──→ [0|0|1]　ページ1のみ表示．

2 ──→ [0|1|0]　ページ2のみ表示．

3 ──→ [0|1|1]　ページ1とページ2を合成して表示．

4 ──→ [1|0|0]　ページ3のみ表示．

5 ──→ [1|0|1]　ページ1とページ3′を合成して表示．

6 ──→ [1|1|0]　ページ2とページ3を合成して表示．

7 ──→ [1|1|1]　全ページを合成して表示．

この〈アクティブページ〉と〈ディスプレイページ〉の設定は気を付けないと，間違ってページを設定したり，書き込んだはずなのに表示ページになっていないなどのため，思った通りに表示できないことがあります．

SCREEN 命令を実行すると現在設定されているウィンドウ，ビューポート，LP は初期状態に戻されます（LP の初期状態はワールド座標，スクリーン座標とも（0，0）です）．

注意：白黒モード（高分解能白黒モードを含む）において，グラフィック命令で〈パレット番号〉を指定した場合は，0か0以外かによって黒か白かを判定します．また〈パレット番号〉を省略した場合は，カラーモードと同様，現在 COLOR 文によって設定されている〈フォアグラウンドカラー〉のパレット番号が採用されます．その際も0か0以外かの判定により白，黒を決定します．

注意：次の場合，SCREEN文のパラメータ4によって指定した表示画面の設定はリセットされます.（3画面が合成されて表示されます.）

・WIDTH文の実行

・テキスト画面のカラー/モノクロモードの切替（CONSOLE文）

・CMD TEXT ON文で，テキスト画面を再表示させた場合（CMD SING "X1", CMD CUTも含む）.

このような場合には，各命令の実行直後，SCREEN1を実行してグラフィックモードの再設定を行ってください.

サンプルプログラム

```
100 SCREEN 0,0:GOSUB *DISPLAY
110 SCREEN 1,0,0,7:GOSUB *DISPLAY
120 SCREEN 2,0:GOSUB *DISPLAY
130 '
140 SCREEN 0,1:GOSUB *DISPLAY
150 SCREEN 0,2:GOSUB *DISPLAY
160 SCREEN 0,0
170 END
180 *DISPLAY
190  FOR I=0 TO 6
200   LINE(I*90,0)-STEP(89,199),I+1,BF
210  NEXT I:CLS 2
220 RETURN
```

```
100 SCREEN 0,0:GOSUB *GRPINFO
110 VIEW(0,0)-(320,100)
120 WINDOW(-100,-100)-(100,100)
130 POINT (50,50)
140 GOSUB *GRPINFO
150 END
160 *GRPINFO '-- graphic information --
170  PRINT "View    :";
180  FOR I=0 TO 3:PRINT USING "#####";VIEW(I);
190  NEXT I:PRINT
200  PRINT "Window :";
210  FOR I=0 TO 3:PRINT USING "#####";WINDOW(I);
220  NEXT I:PRINT
230  PRINT "LP      :";
240  FOR I=0 TO 3:PRINT USING "#####";POINT(I);
250  NEXT I:PRINT:PRINT:RETURN

run
View    :    0    0  639  199
Window  :    0    0  639  199
LP      :    0    0    0    0

View    :    0    0  320  100
Window  : -100 -100  100  100
LP      :   50   50  240   75

Ok
```

SET (ディスク)　N88 N

機　能　ファイル属性のセット・リセット

書　式　1) SET 〈ドライブ番号〉, 〈"属性文字"〉

2) SET 〈ファイル名〉, 〈"属性文字"〉

3) SET #〈ファイル番号〉, 〈"属性文字"〉

文　例　1) SET 1, "P"

2) SET "2 : NOTWRITE", "P"

3) SET #1, "R"

解　説　指定したドライブ，ファイル番号，ファイルに，〈"属性文字"〉によって指定された属性を付けます。属性文字は，P，R(大文字)で指定し，それぞれ，ライトプロテクト(書き込み禁止)，リードアフターライト(書き込み確認)の属性を付けます。これ以外の文字が属性文字として指定された場合には，現在設定されている属性が全て解除されます。

1) 〈ドライブ番号〉が指定された場合には，指定されたドライブの中に入っているディスケットに対して属性がつけられます。

2) 〈ファイル名〉が指定された場合には，指定されたファイルのみに属性がつけられ同一ディスケット中の他のファイルにはその影響は及びません。

3) 〈ファイル番号〉が指定された場合には，以後そのファイルへの出力に対して指定された属性が作用します。

P属性を付けると，PRINT #，PUT，WRITE #等の書き込み動作が禁止されるとともに，ファイルのKILLもできなくなります。

R属性を付けると，書き込みを行った直後に読み出しを行い正しく書き込みが行れたかの確認がされるようになります。

注意：P属性をつけても，APPENDモードでファイルをオープンすれば，PRINT #，WRITE #は実行できます。

参照：ATTR$

サンプル
プログラム

```
save "1:sample"
Ok
set 1,"P"
Ok
kill "1:sample"
File write protected
Ok
```

STOP　N88　N

機　能	プログラムの実行を停止してコマンドレベルに戻る.
書　式	STOP
文　例	STOP

解　説　STOP文はプログラムの実行を停止するためのもので，プログラムのどこで使用してもかまいません．STOP文を実行すると，次のメッセージがプリントされます．

BREAK in lllll　　(lllllはストップした行番号)

END文と異なり，STOP文はファイルを閉じません．

STOPが実行されると，BASICは常にコマンドレベルに戻ります．また，CONTコマンドによりプログラムの実行は再開されます．但し，INPUT文などの入力文の途中でストップした場合，継続することはできません．

参照：CONT

サンプルプログラム

```
list
100 FOR I=10 TO 19
110  PRINT I,SQR(I)
120  STOP
130 NEXT I
140 END
Ok
run
 10             3.16228
Break in 120
Ok
print sqr(i)*sqr(i)
 10
Ok
cont
 11             3.31662
Break in 120
Ok
```

STOP ON/OFF/STOP　N88

機　能	ストップキーによる割り込みの許可，禁止，停止を設定する．
書　式	1) STOP　ON 2) STOP　OFF 3) STOP　STOP
文　例	STOP ON

解　説

1）割り込みを許可します．以後ストップキーが押されるごとにON　STOP GOSUB文によって設定された処理ルーチンに分岐します．

2）割り込みを禁止します．以後ストップキーを押しても割り込み処理ルーチンへの分岐は起こりません．

3）割り込みを停止します．以後ストップキーを押してもその事を覚えているだけで，処理ルーチンへの分岐は起こりません．しかしSTOP　ON命令によって割り込みが許可されると，先程のストップキーを押したことによる割り込みで処理ルーチンに分岐します．

注意：プログラムの最後では必ずSTOP　OFFを実行してください．もし怠ると本来のストップキーとしての機能を失います．

参照：ON STOP GOSUB，KEY ON/OFF/STOP

サンプルプログラム

```
stop on

stop off

stop stop
```

SWAP　N88　N

機　能　　2つの変数の値を交換する.

書　式　　SWAP <変数>, <変数>

文　例　　SWAP　A$, B$

解　説　どの型の変数でも,SWAP文によりその値を交換することができます(整数, 単精度, 倍精度, 文字, または配列変数). しかしその2つの変数の型は一致していないと〝Type　mismatch〞(? TM Error) エラーが起こります.

サンプルプログラム

```
100 ' 2ﾂ ﾉ ﾊｲﾚﾂ ﾉ ﾅｲﾖｳ ｦ ｲﾚｶｴﾙ
110  DIM A(10),B(10)
120  FOR I=1 TO 10
130   A(I)=I*2:B(I)=I*3:GOSUB *PRINTAB
140  NEXT I:LOCATE 0,Y+3
150  FOR I=1 TO 10
160   SWAP A(I),B(I):GOSUB *PRINTAB
170  NEXT I:LOCATE 0,Y+2
180 END
190 *PRINTAB
200  X=POS(0):Y=CSRLIN
210  LOCATE X,Y:PRINT USING "###";A(I);
220  LOCATE X,Y+1:PRINT USING "###";B(I);
230  LOCATE X+5,Y
240 RETURN

run
  2    4    6    8   10   12   14   16   18   20
  3    6    9   12   15   18   21   24   27   30

  3    6    9   12   15   18   21   24   27   30
  2    4    6    8   10   12   14   16   18   20
Ok
```

TERM　N88

機　能　システムをターミナルモードにする.

書　式　TERM "〔COM : 〕〔〈パリティ〉〔〈ビット〉〔〈ストップビット〉〔〈XON スイッチ〉〔〈S パラメータ〉〕〕〕〕" 〔, 〔〈モード〉〕〔, 〈変数領域の大きさ〉〕〕

文　例　TERM "COM : E72X",, 1000

解　説　PC-8801MKII をBASICモードから，ターミナルモードに変化させます。ターミナルモードでは，RS-232Cインターフェイスを介して他のコンピュータや機器との通信が可能となります．また，ターミナルモード時においても，リモート BASIC プロトコルにより，N_{88}-BASIC の機能を利用することができます．詳しくは，ユーザーズ・マニュアルを参照してください．

最初の COM : でコミュニケーションポートを使って外部機器との入出力を行うことを宣言します．

また COM : を省略した場合は，後に続くすべてのパラメータを指定しなければなりません．

〈パリティ〉はE，O，Nで表わし，それぞれ偶数パリティ，奇数パリティ，パリティ無しを意味します．

〈ビット〉は，1文字を表わすのに用いるビット数で，7または8で指定します．7を指定した場合には，必ず〈パリティ〉でOまたはEを指定しなければなりません．また8を指定した場合には〈パリティ〉はNでなければなりません．

〈ストップビット〉は，ストップビット数を決定し，1，2，3で指定します．1は1ビット，2は1.5ビット，3は2ビットを意味します．

〈XON スイッチ〉は，XON/XOFF による通信制御を行うことを指定し，スイッチとしてXが指定されると XON/XOFF 制御を行います．Nが指定された場合にはこの制御は行われません．

〈Sパラメータ〉は〈ビット〉に7を指定した時に，キャラクタコード128以上の文字（カナ文字など）の入出力ができなくなるのを補助するパラメータです．指定はSまたはNで行い，Sを指定した時入出力可能で，Nを指定した時は入出力不可能になります．

〈モード〉は，HまたはFで指定し，それぞれ半二重，全二重通信のモードを意味します．

〈変数領域の大きさ〉は，リモート BASIC プロトコルを利用する際に用いられ

る変数領域の大きさを決定します．省略された場合1024に設定されます．

ターミナルモードからBASICモードに戻る時には，シフトキーとストップキーを同時に押します．

ダブルクオーツ（″）で囲まれた部分（ダブルクオーツを含む）は，OPENによりRS‐232Cコミュニケーション・ファイルを開く場合にも同一の書式で用いられます．

注意：〈変数領域の大きさ〉を除くすべてのパラメータは，省略された場合，リセットした時（電源投入時も同様）に設定されていたディップスイッチの値が採用されます（ディップスイッチについては，ユーザース・マニュアルを参照して下さい）．また途中のパラメータを省略して，後のパラメータを指定する場合は，省略記号としてブランク（〝　″）を指定しなければなりません．

ここで使用する各パラメータは，必ず大文字を使って下さい．

参照：ユーザース・マニュアル

サンプルプログラム

```
term "COM:N8 20
term "COM:E72 S",,1000
```

TERM　Ⓝ

機　能	**システムをターミナルモードにします。**
書　式	**TERM 〈データのビット長〉, 〈パリティ〉, 〈ボーレート〉, 〈フィードスイッチ〉**
文　例	**TERM J, 2, 1, 1**

解　説　N-BASICモードからターミナルモードに変化させます。

ターミナルモードでは，RS-232Cインターフェイスを介して，他のコンピュータや周辺機器との通信が可能となります．

〈データのビット長〉，〈パリティ〉，〈ボーレート〉，〈フィードスイッチ〉は次のようになっています．

〈データのビット長〉	a：8ビットモード j：7ビットモード
〈パリティ〉	0：パリティなし 1：奇数パリティ 2：偶数パリティ
〈ボーレートファクタ〉	0：×64 1：×16
〈フィードスイッチ〉	0：オートL/Fしない 1：オートL/Fする

〈データのビット長〉は8ビットモードの場合最上位ビットでカナと英文字の切替えを行い，7ビットモードでは，SI(shift in), SO(shift out)コードで英文字とカナ文字の切り替えが行われます．

〈パリティ〉はパリティの形式，有無を指定します．

〈ボーレートファクタ〉と，本体前面のジャンパスイッチによって実際のボーレートを決定します．ボーレートファクタが×16のとき前面ジャンパスイッチの値がそのまま採用されます。×64のときはジャンパスイッチの値の4分の1が実際のボーレートとなります．

〈フィードスイッチ〉の指定を1にするとCRコードを受信したときにCR,Fを行い改行します．0のときはCR, LFコードがそれぞれ独立して働きます。

ターミナルモードではファンクションキーの6～10が次のような意味を持っています。

f･6：コントロールコードの表示をする/しない

f･7：キーボードからの入力を表示する/表示しない

f･8：LPに出力する/しない

f･9：画面の文字をプリンタへ出力する

f･10：LPのラインフィードを行う

f･6～f･8は1度押すと現在の逆の状態になります．

ターミナルモードに入った直後はf･6～f･8は次のように設定されています．

f･6：コントロールコードの表示をしない

f･7：キーボードからの入力を表示しない

f･8：LPに出力しない

GRPH-Bで，ターミナルモードからBASICに戻ります．

参照：ユーザーズマニュアル

サンプル
プログラム

```
term j,2,1,1
```

TIME$ ON／OFF／STOP　N88

機　能	リアルタイムタイマによる割り込みの許可，禁止，停止を設定する.
書　式	1) TIME$ ON 2) TIME$ OFF 3) TIME$ STOP
文　例	TIME$ ON

解　説

1) 割り込み動作を許可します．この命令実行後は，設定された時刻になると割り込みが発生し，ON TIME$ GOSUB 文によって定義された処理ルーチンに分岐します．
2) 割り込み動作を禁止します．この命令実行後は，割り込みが発生しません．
3) 割り込み動作を停止します．この命令実行後は，割り込みが発生したことを覚えているだけで，処理ルーチンへの分岐は起こりません．しかし，TIME$ ON 命令により割り込みが許可されると，先程の割り込みによって処理ルーチンへ分岐します．

注意：プログラム終了時には TIME$ OFF を実行して下さい．

ON　TIME＄　GOSUBで"19：00：00以降に割り込みを設定した場合は，割り込みが起きない場合があります．

参照：ON TIME$ GOSUB，TIME$

サンプルプログラム

```
time$ on

time$ off

time$ stop
```

TRON／TROFF　N88 N

機　能	プログラムの実行状態を追跡する.
書　式	TRON TROFF
文　例	TRON TROFF

解　説　プログラムのデバッグの助けとして，TRON 文を実行すると（ダイレクトモードでもプログラムモードでも実行できます）トレースフラグが立ち，実行したプログラム行の行番号を表示します．それぞれの行番号は角カッコに囲んで表示します．トレースフラグは TROFF 文（または NEW コマンド）を実行するとリセットされます．

サンプルプログラム

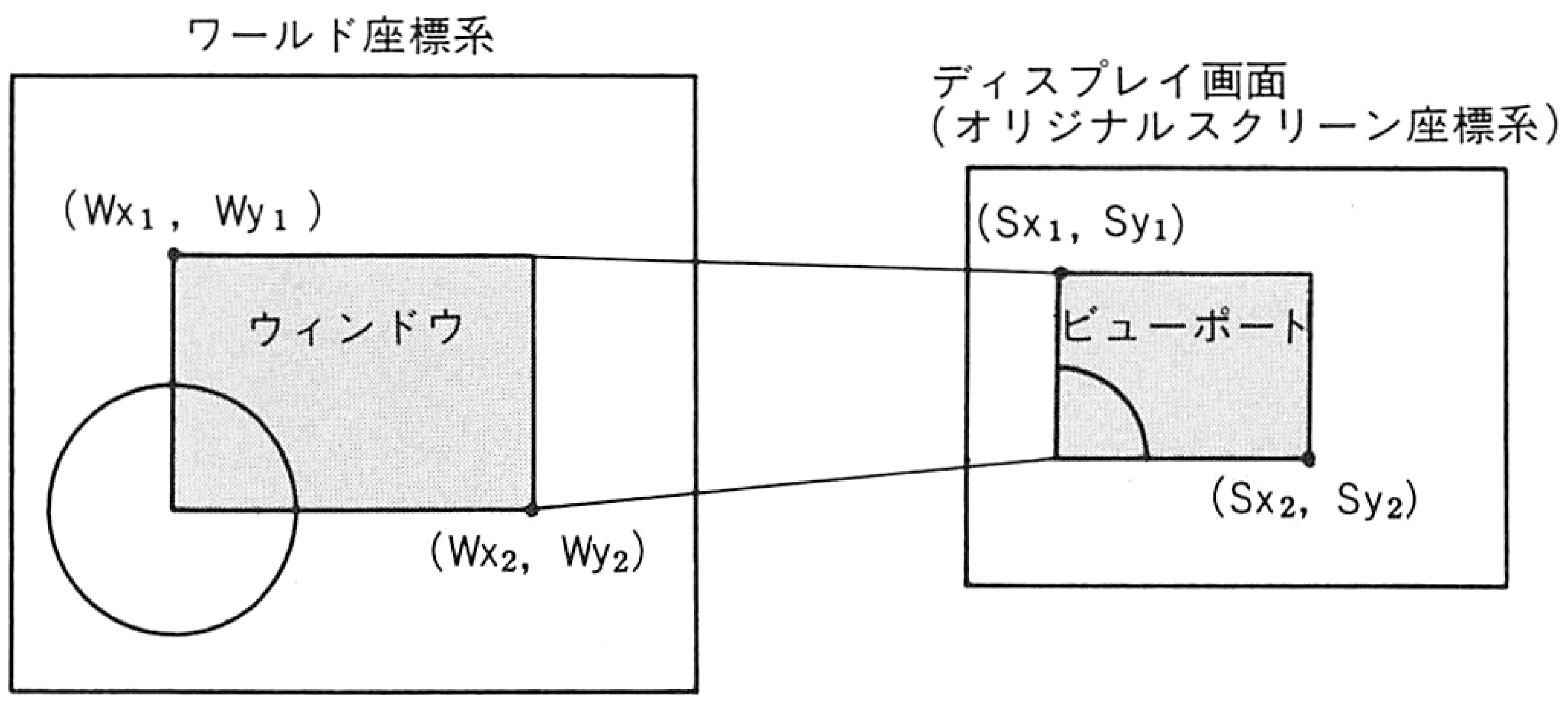
```
10 FOR I=1 TO 3
20  PRINT I,
30 NEXT I

tron
Ok
run
[10][20] 1     [30][20] 2
[30][20] 3     [30]
Ok
troff
Ok
run
 1              2
 3
Ok
```

VIEW　N88

機　能	ディスプレイ画面上での表示領域（ビューポート）を指定する.
書　式	VIEW（Sx_1，Sy_1）−（Sx_2，Sy_2）〔，<領域色>〕〔，<境界色>〕
文　例	VIEW(100,30)−(200,75),,7

解　説　（Sx_1，Sy_1）を左上の頂点，（Sx_2，Sy_2）を右下の頂点とするオリジナルスクリーン座標系上の長方形を図形表示領域として指定します．ここでいうオリジナルスクリーン座標とは，ビューポート内に展開されるスクリーン座標ではなく，ディスプレイ画面のドットと1対1に対応して，WINDOW，VIEWにより変化することのない物理的な座標のことです．以後，WINDOW により指定されているワールド座標系上の領域内に入る図形は，VIEW で指定された領域中に表示されるようになります．この領域はビューポート（View Port）と呼ばれます．



<領域色>が指定された場合には，指定されたパレット番号でビューポート内を塗りつぶします．

<境界色>が指定された場合には，指定されたパレット番号でビューポートの枠を描きます．CLS によって消去される画面の範囲は，この枠によって囲まれた中だけで，枠は消されることはありません．

VIEW は，図形の表示領域の指定を行うだけで，実際の画面に対する操作は行いませんので，VIEW によりビューポートを変更するだけで，以前のビューポート中の図形が移動するようなことはありません．また，VIEW はすべてのグラフィック画面表示の対象範囲を指定された領域に限定してしまいますから，ビューポートの外に点や線等を表示することはできません．

VIEW の指定を変えるだけで，1つの図形を描くルーチンでも，画面上の異な

V

る位置に異なる大きさで描くようになります.

$Sx_1 < Sx_2$, $Sy_1 < Sy_2$ が成り立たない場合，あるいはこれらの座標がディスプレイ画面から外れている場合には，〝Illegal function call〟(FC Error) エラーとなります.

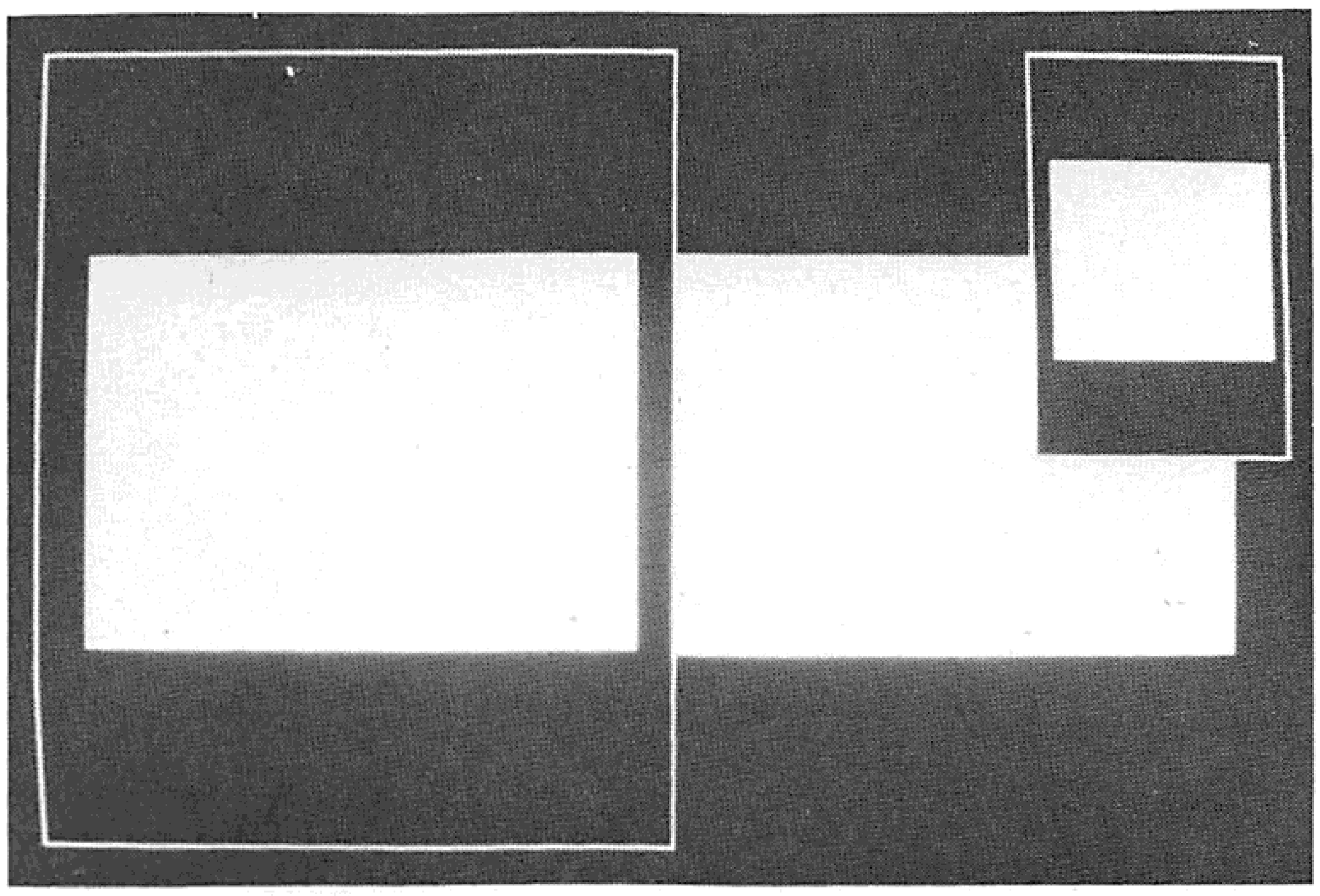
一度設定されたビューポートは，次に VIEW 文あるいは SCREEN 文が用いられるまで変化しません. また VIEW は LP をビューポートの左上の頂点に設定します.

注意：ワールド座標系がビューポート内に展開されるのは，WINDOW 文の実行後であり，VIEW のみ実行してウィンドウは初期状態のままであれば，ビューポート内の座標は，(ワールド座標) = (スクリーン座標) となります.

参照：WINDOW，CLS，SCREEN，1 章・ウィンドウとビューポート，APPENDIX・A

サンプルプログラム

```
100 SCREEN 0,0:CLS 2
110 WINDOW(0,0)-(639,199)
120 GOSUB *DRAW
130 VIEW(1,1)-(319,198),0,7
140 GOSUB *DRAW
150 VIEW(500,1)-(630,99),0,7
160 GOSUB *DRAW
170 END
180 *DRAW
190  LINE(50,50)-(600,150),7,BF
200 RETURN
```



WAIT　N88　N

機　能　コンピュータの入力ポートをモニタする間，プログラムの実行を停止する．

書　式　WAIT 〈ポート番号〉，〈式 1〉〔，〈式 2〉〕

文　例　WAIT 1，&H22，&H22

解　説　WAIT 文を実行すると，指定した入力ポートのビットパターンが指定した状態になるまでプログラムの実行が中断されます．指定したポートから読み込んだデータと〈式 2〉との XOR の結果と〈式 1〉の AND が取られます．もしその結果が 0（偽）なら，BASIC はもう一度ポートの状態を読み込み同じ操作を繰り返します．もし結果が 0 でなければ(真)，プログラムの実行は次の文に移ります．〈式 2〉が省略された場合は 0 と見なします．

注意：WAIT 文の実行により無限ループにはいってしまう場合があります．その場合にはコンピュータをリセットしなければなりません．

サンプルプログラム

```
100 '-- read from key port --
110 PRINT "Type 's' to start."
120 WAIT 4,8,255
130 '-- START --
140 END

run
Type 's' to start.
Ok
s
```

WHILE～WEND（ディスク） N88

機　能	条件が満足されている間繰り返し実行する.
書　式	WHILE 〈論理式〉 ～ WEND
文　例	WHILE J< 5 ～ WEND

解　説　〈論理式〉が真（0でない）である間，WHILE文とWEND文の間にある文の実行を繰り返します．

〈論理式〉を評価した結果が真であれば，WHILE文とWEND文との間にある文を順番に実行し，WHILE文に戻ります．この繰り返しは，〈論理式〉の評価が偽になるまで続けられ，〈論理式〉が偽になると，WEND文に続く文に制御が移ります．最初に〈論理式〉を評価した結果が偽であれば，WHILE文とWEND文との間にある文は一度も実行されません．

WHILE～WEND文はFOR～NEXT文と同じ様に入れ子構造にすることができます．この場合には，それぞれのWEND文は，それ以前の最も近くにあるWHILE文と対応すると解釈されます．WEND文は，WHILE文と対になり，ループの終りを示す働きをしますので省略することはできません．

サンプルプログラム

```
100 I=1
110 WHILE I<6
120  PRINT I;"-";
130  J=1
140  WHILE J=<I
150   PRINT J;:J=J+1
160  WEND
170  PRINT:I=I+1
180 WEND

run
 1 - 1
 2 - 1  2
 3 - 1  2  3
 4 - 1  2  3  4
 5 - 1  2  3  4  5
Ok
```

WIDTH　N88

機能	デバイスやファイルに対して文字の領域を設定する.
書式	1) WIDTH 〈桁数〉〔,〈行数〉〕
	2) WIDTH 〈ファイルディスクリプタ〉,〈サイズ〉
	3) WIDTH #〈ファイル番号〉,〈サイズ〉
文例	1) WIDTH 80, 25
	2) WIDTH "LPT1 :", 80
	3) WIDTH #1, 100

解説

1) テキスト画面に表示する文字の行数と桁数を設定します.

1行あたり40文字, または80文字の〈桁数〉を設定できます. また〈行数〉は20行から25行のどちらかです. もし〈行数〉を省略した場合は現在の行数のままで変化しません. また, この文を実行すると画面がクリアされ, スクロールウィンドウが初期化されます. WIDTH文はCONSOLE文の前に実行してください.

2)〈ファイルディスクリプタ〉で指定されたデバイスに対してバッファのサイズを設定します. ここで許されているデバイスは, コミュニケーションポートとプリンタです.〈サイズ〉の値は0から255まで許されており0は256と解釈されます. たとえば, この命令がプリンタに対して実行されたとすれば, プリンタの1行に印字される文字数は〈サイズ〉で指定したものとなります. このようにプリンタに対して実行した場合, WIDTH LPRINTと同等の機能を得ることができます. 初期状態では255に設定されています.

3)〈ファイル番号〉で割り当てられているバッファ(チャネル)に対して, その大きさを〈サイズ〉で指定します. サイズは0から255までの値で0は256と解釈されます. 以後このファイル番号を使っているデバイスとの入出力はこの〈サイズ〉単位で行われます. 初期状態では255に設定されています.

注意:WIDTH文を使って, テキスト画面の表示文字数を変更すると, SCREEN文のパラメータ4がリセットされます.(SCREEN文参照).

参照:WIDTH LPRINT

サンプルプログラム

```
width 80,25
width 40,25
width 80
width 40,20
```

```
width "lpt:",40
width "com:",255
```

```
100 OPEN "lpt:" FOR OUTPUT AS #1
110 WIDTH #1,20
120 FOR I=1 TO 100
130  PRINT #1,I;
140 NEXT I
150 PRINT #1,
160 END
```

```
 1  2  3  4  5  6  7
  8  9  10  11  12
13  14  15  16  17
18  19  20  21  22
23  24  25  26  27
28  29  30  31  32
33  34  35  36  37
38  39  40  41  42
43  44  45  46  47
48  49  50  51  52
53  54  55  56  57
58  59  60  61  62
63  64  65  66  67
68  69  70  71  72
73  74  75  76  77
78  79  80  81  82
83  84  85  86  87
88  89  90  91  92
93  94  95  96  97
98  99  100
```

WIDTH　N

機　能　画面に表示する文字の行数と桁数を指定する.

書　式　WIDTH〔<桁数>〕〔,<行数>〕

文　例　WIDTH 80, 25

解　説　画面に表示する文字の行数と1行の桁数を設定します.

N_{88}-BASICのWIDTHの1)と同じ使い方ですが, N-BASICの場合, 1行36桁と72桁の表示も許されます. また最初の〈桁数〉も省略することができます. 省略値は現在の桁数です.

参照：N_{88} WIDTH　1)

サンプルプログラム

```
width 80,25
width ,20
width 40,20
width 72,25
width 36,20
width 80
```

WIDTH LPRINT　N₈₈

機　能	プリンタに出力される1行の表示文字数を設定する.
書　式	WIDTH LPRINT 〈文字数〉
文　例	WIDTH LPRINT 80

解　説　プリンタの1行に印字することのできる文字数を設定します.〈文字数〉で指定することのできる値は，0から255までで0は256と解釈されます.

参照：N_{88} WIDTH　2)

サンプルプログラム

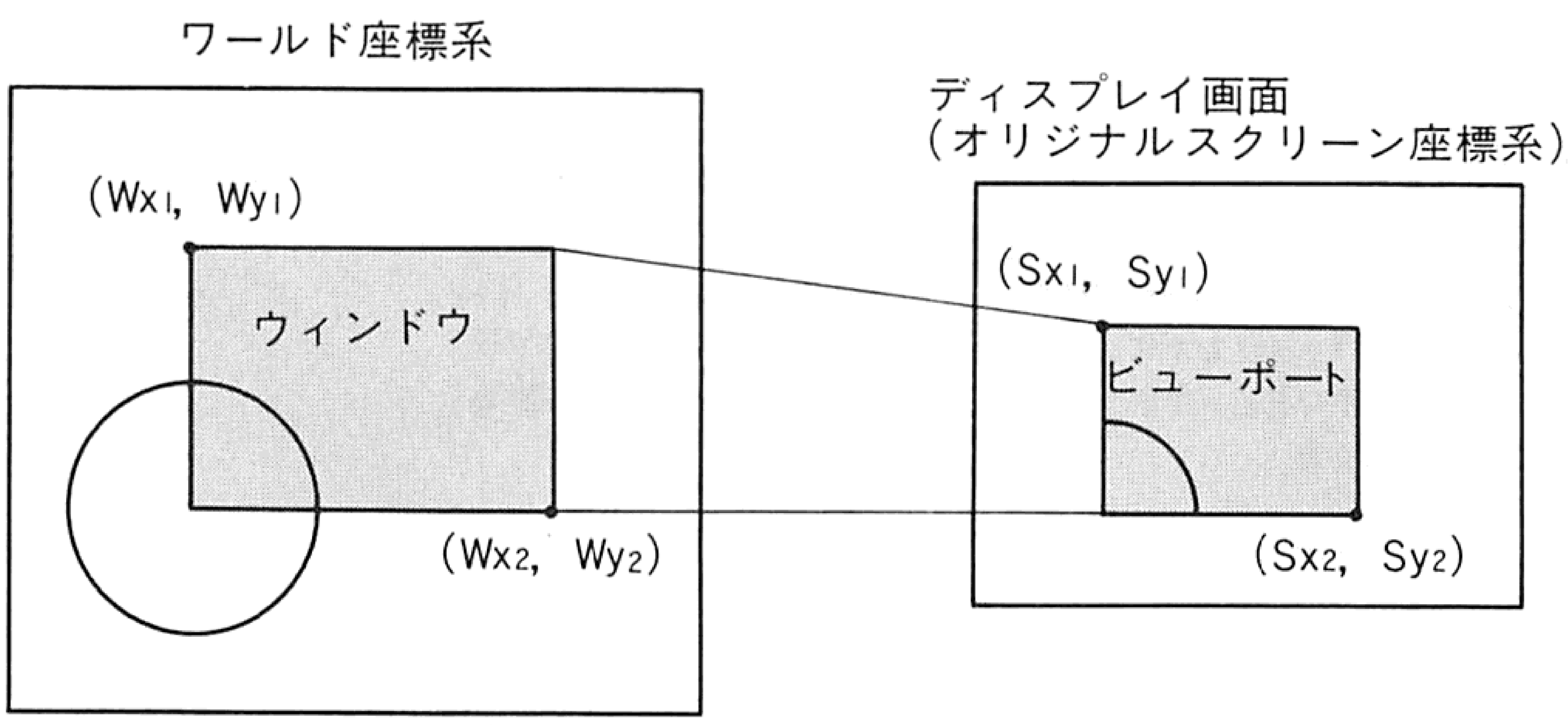
```
100 WIDTH "lpt:",255
110 WIDTH LPRINT 40
120 FOR I=33 TO 247
130  LPRINT CHR$(I);
140 NEXT I
150 LPRINT:END

run
!"#$%&'()*+,-./0.123456789:;<=>?@ABCDEFGH
IJKLMNOPQRSTUVWXYZ[¥]^_`abcdefghijklmnop
qrstuvwxyz{¦}~ ▁▂▃▄▅▆▇█▏▎▍▌▋▊▉┼┴┬┤├▔─│▕┌
┐└┘╭╮╰╯ 。「」、・ヲァィゥェォャュョッーアイウエオカキクケコサシスセソタ
チツテトナニヌネノハヒフヘホマミムメモヤユヨラリルレロワン゛゜═╞╪╡▲▼♠
♥♦♣●○╱╲╳円年月日時分秒
OK
```

WINDOW N88

機　能	**ビューポートに表示するワールド座標系内の領域を指定する.**
書　式	WINDOW (Wx_1, Wy_1) − (Wx_2, Wy_2)
文　例	WINDOW(−300, −50) − (100, 70)

解　説　(Wx_1, Wy_1) を左上の頂点，(Wx_2, Wy_2) を右下の頂点とするワールド座標系上の長方形で囲まれた領域を，スクリーン上のビューポートに表示する領域として指定します．以後，この領域内に入る図形は VIEW で指定された領域（ビューポート）内に表示されるようになります．このようにして指定された領域は，ウィンドウと呼ばれ，ちょうどカメラのファインダを通して景色を見た時の様に，ウィンドウから外れた図形は表示されなくなります．また，ウィンドウの大きさを変えることにより，カメラのズーミングの様に，視野に入る領域の大きさが変り，それがスクリーン上のビューポートに表示されるようになります．



WINDOW も VIEW と同じ様に領域の指定を行うだけで，実際の画面に対する操作は行いませんので，WINDOW により，ウィンドウの設定を変更するだけで，ビューポート中に表示される図形が変化するようなことはありません．また，誤って，図形が何も描かれない領域をウィンドウとして設定すると，スクリーン上のビューポートには，何の図形も描かれませんで注意してください．

$Wx_1 < Wx_2$, $Wy_1 < Wy_2$ が成り立たない場合，あるいはこれらの座標がワールド座標系から外れている場合には "Illegal function call" (FC Error) エラーとなります．WINDOW で指定する座標は，ワールド座標ですから，負の値や実数値もとれることに注意してください．

一度設定されたウィンドウは，次に WINDOW 文あるいは SCREEN 文が用い

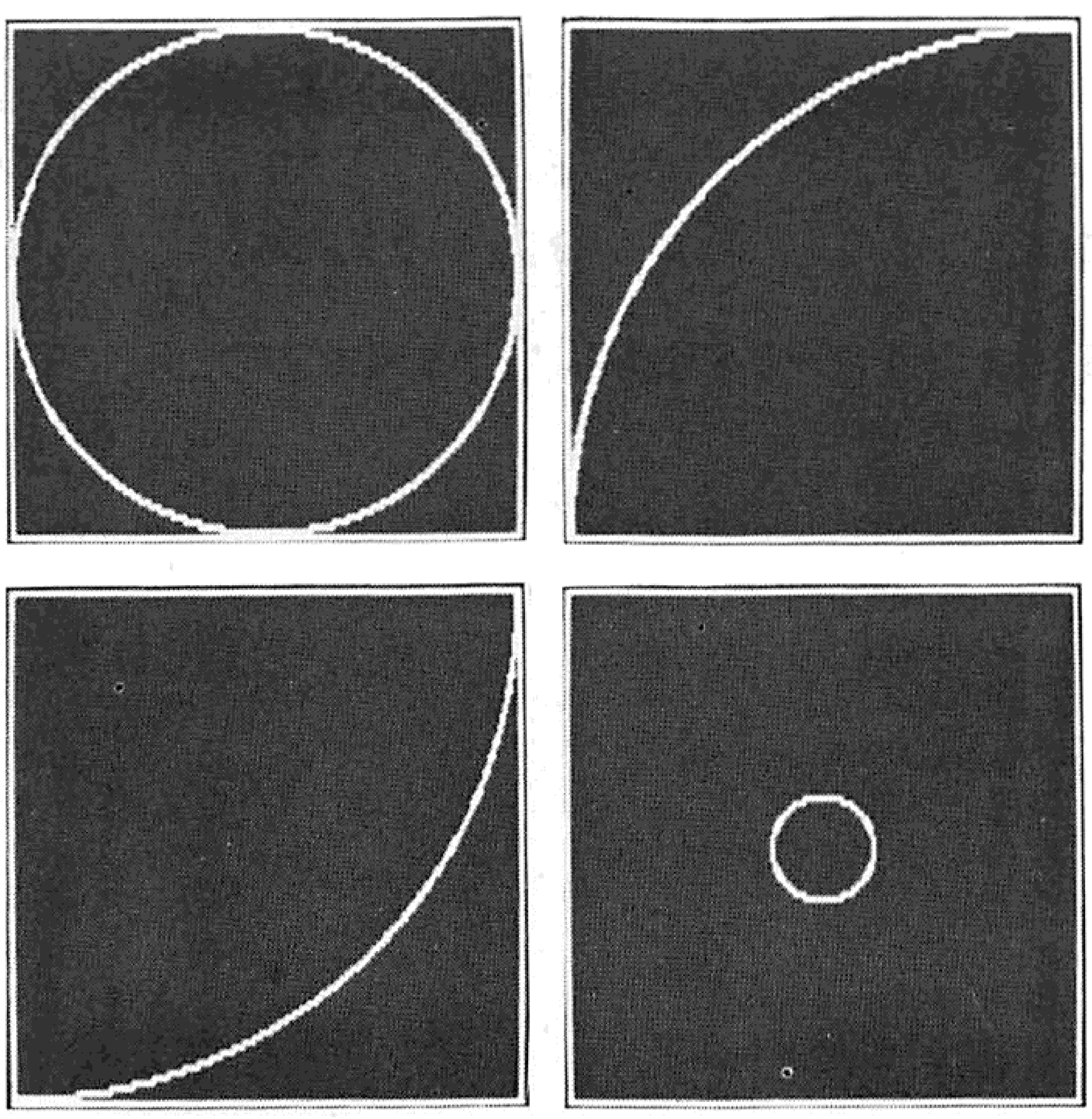
られるまで変化しません．また WINDOW は LP をウィンドウの左上の頂点に設定します．

注意：ワールド座標系がビューポート内に展開されるのは，WINDOW 文の実行後であり，VIEW のみ実行してウィンドウは初期状態のままであれば，ビューポート内の座標は，（ワールド座標）＝（スクリーン座標）となります．

参照：VIEW，1章・ウィンドウとビューポート，APPENDIX・A

サンプルプログラム

```
100 SCREEN 0,0:CLS 2
110 C=1
120 VIEW(200,50)-(400,150),,7
130 WINDOW(-100,-100)-(100,100)
140 GOSUB *DRAW
150 WINDOW(-100,-100)-(0,0)
160 GOSUB *DRAW
170 WINDOW(0,0)-(100,100)
180 GOSUB *DRAW
190 WINDOW(-500,-500)-(500,500)
200 GOSUB *DRAW
210 END
220 *DRAW
230  CIRCLE(0,0),100,C
240  FOR J=0 TO 1000:NEXT J
250  C=C+1:CLS 2
260 RETURN
```



WRITE（ディスク） N88

機　能	画面へデータを出力する.
書　式	WRITE〔<式の列>〕
文　例	WRITE　I＊J, K；A$

解　説　<式の列>により指定された，数値式あるいは文字式の値を画面に出出します．<式の列>が省略された場合には空行のみが出力されます．式は，コンマ（，）あるいはセミコロン（；）により区切ります．コンマとセミコロンとの機能上の区別はありませんが，画面上には，各式の値の区切りとして必ずコンマが表示されます．BASICは，各式の値を出力したのち，改行を行います．

WRITE文は，ほぼPRINT文と同じ様に式の値の出力を行いますが，不要な空白桁は詰められ，それぞれの式の値の間は必ずコンマで区切られます．また文字列はダブルクォーツ（″）で囲まれて出力されます．

参照：PRINT，WRITE#

サンプルプログラム

```
100 A%=123
110 B!=9.87654E+31
120 C#=3.14159265359#
130 D$="N88-BASIC"
140 PRINT A%;B!,C#;D$
150 WRITE A%;B!,C#;D$
```

```
run

 123  9.87654E+31              3.141592653
59 N88-BASIC
123,9.87654E+31,3.14159265359,"N88-BASIC
"
Ok
```

WRITE #（ディスク） N88

機　能	シーケンシャルファイルへデータを書き出す．
書　式	WRITE #〈ファイル番号〉〔，〈式の列〉〕
文　例	WRITE #1, I * J, K ; A$

解　説　〈式の列〉により指定された数値式あるいは文字式の値を，〈ファイル番号〉により指定されたファイルに書き出します．指定されたファイルは出力用にオープンされていなければなりません．〈式の列〉中の各式は，コンマ（,）あるいは，セミコロン（；）によって区切ることができます．

WRITE #は，ほぼPRINT #と同じように式の値を出力しますが，不要な空白桁は詰められ，それぞれの式の値の間は必ずコンマで区切られます．また文字列はダブルクォーツ（″）で囲まれて出力されます．

WRITE #は，各式の値を書き出した後，改行のコードを書き出します．〈式の列〉が省略された場合には，単に改行のコードのみが書き出されます．

WRITE #は，区切り記号としてコンマを必ず出力し，また不要な空白の出力は行わないため，PRINT #に比べファイルの使用領域を節約することができます．

参照：PRINT #，WRITE

サンプルプログラム

```
100 '-- write --
110 OPEN "data5" FOR OUTPUT AS #1
120 INPUT "ﾅﾏｴ ?",NA$
130 INPUT "ｼｭｯｾｷ ﾊﾞﾝｺﾞｳ";NM
140 INPUT "ﾃﾝｽｳ";PO
150 WRITE #1,NA$,NM,PO
160 CLOSE

run
ﾅﾏｴ ?ﾀﾅｶ ﾋﾛｼ
ｼｭｯｾｷ ﾊﾞﾝｺﾞｳ? 18
ﾃﾝｽｳ? 98
Ok
```

第3章

関　　数

ABS　N88 N

機　能	絶対値を与える.
書　式	ABS (〈数式〉)
文　例	B=ABS(－2)

解　説　〈数式〉の絶対値を返します.

また, 〈数式〉に倍精度実数が含まれる場合は, 倍精度の値を与えますが, 他の場合には, 単精度の値を与えます.

サンプルプログラム

```
100 '-- 2ﾃﾝｶﾝ ﾉ ｷｮﾘｦ ﾓﾄﾒﾙ --
110 INPUT "ﾃﾝ1 ﾉ ｻﾞﾋｮｳ ﾊ: x,y ";X1,Y1
120 INPUT "ﾃﾝ2 ﾉ ｻﾞﾋｮｳ ﾊ: x,y ";X2,Y2
130 D=SQR(ABS(X1-X2)^2+ABS(Y1-Y2)^2)
140 PRINT:PRINT "ﾃﾝ1 ﾄ ﾃﾝ2 ﾉ ｷｮﾘ ﾊ ";D
run
ﾃﾝ1 ﾉ ｻﾞﾋｮｳ ﾊ: x,y ? 0,0
ﾃﾝ2 ﾉ ｻﾞﾋｮｳ ﾊ: x,y ? 100,0

ﾃﾝ1 ﾄ ﾃﾝ2 ﾉ ｷｮﾘ ﾊ  100
OK
```

ASC　N88　N

機　能	**文字のキャラクタコードを与える.**
書　式	ASC（〈文字列〉）
文　例	A＝ASC（"A"）

解　説　〈文字列〉の最初の文字のキャラクタコードを与えます．文字とキャラクタコードの対応については，キャラクタコード表を参照してください．

参照：CHR$，APPENDIX・キャラクタ

サンプルプログラム

```
100 '-- ｺﾓｼﾞ --> ｵｵﾓｼﾞ ﾍﾝｶﾝ --
110 INPUT A$
120 IF A$="" THEN 200
130 B$=""
140 FOR I=1 TO LEN(A$)
150  D$=MID$(A$,I,1):D=ASC(D$)
160  IF D<97 OR D>122 THEN B$=B$+D$:GOTO 180
170  B$=B$+CHR$(D-32)
180 NEXT I
190 PRINT:PRINT B$
```

```
run
? nec pc-8801mk2

NEC PC-8801MK2
Ok
```

ATN (ディスク)　N88 N

機　能	逆正接（アークタンジェント）を与える.
書　式	ATN（〈数式〉）
文　例	A=ATN(1)

解　説　〈数式〉の逆正接をラジアンで与えます. 得られる値は, $-\pi/2$ から $\pi/2$ までの範囲です.

また, 〈数式〉に倍精度実数が含まれる場合は, 倍精度の値を与えますが, 他の場合には, 単精度の値を与えます.

サンプルプログラム

```
100 '-- arctan ｶﾗ arcsin,arccos ｦ ﾓﾄﾒﾙ
110 T=180/3.14159:U1$="#.###":U2$="##.####     ###.##"
120 INPUT A
130 IF ABS(A)>1 THEN 120
140 IF ABS(A)=1 THEN AS=1.5708:AC=0:GOTO 170
150 AS=ATN(A/SQR(-A*A+1))
160 AC=-ATN(A/SQR(-A*A+1))+3.14159/2
170 DS=AS*T:DC=AC*T
180 PRINT "                    Rad.       Deg."
190 PRINT "arcCOS(";USING U1$;A;:PRINT ") =";USING U2$;AC;DC
200 PRINT "arcSIN(";USING U1$;A;:PRINT ") =";USING U2$;AS;DS
210 END
run
? .5
                    Rad.       Deg.
arcCOS(0.500) = 1.0472      60.00
arcSIN(0.500) = 0.5236      30.00
Ok
```

ATTR$ (ディスク)　N88 N

機能　指定ファイル，ドライブの属性を与える．

書式　ATTR$ | (<ドライブ番号>) / (# <ファイル番号>) / (<ファイル名>) |

文例　PRINT ATTR$(1)

解説　指定したドライブ，オープンされているファイル，ディスク上のファイルの現在の属性文字を与えます．この時，属性文字は以下の意味を持ちます．

〝□□□″：属性は解除されています．即ち，通常のリード，ライトが可能な状態です．

〝R″：書き込みの際に，リードアフターライト（書き込んだ内容とメモリ上の内容の比較チェック）を行います。

〝□□P″：書き込みが禁止されています．

参照：SET

サンプルプログラム

```
100 REM ﾃﾞｨｽｸｯﾄ ﾄ ﾌｧｲﾙ ﾉ ｿﾞｸｾｲ ｦ ﾁｪｯｸ ｽﾙ
110 INPUT "ﾌｧｲﾙ ﾒｲ : ";F$
120 IF MID$(F$,2,1)<>":" THEN 150
130 DR=VAL(F$):F1$=MID$(F$,3)
140 GOTO 160
150 DR=1:F1$=F$
160 AT$=ATTR$(DR)
170 PRINT "ﾄﾞﾗｲﾌﾞ";STR$(DR);" ﾉ ｿﾞｸｾｲ ﾊ : ";:GOSUB 220
180 IF MID$(F$,3)="" THEN 210
190 AT$=ATTR$(F$)
200 PRINT "ﾌｧｲﾙ ";F1$;" ﾉ ｿﾞｸｾｲ ﾊ : ";:GOSUB 220
210 END
220 A$="ｶｲｼﾞｮ"
230 IF AT$="R  " THEN A$="ﾘｰﾄﾞ ｱﾌﾀｰ ﾗｲﾄ"
240 IF AT$="  P" THEN A$="ﾗｲﾄ ﾌﾟﾛﾃｸﾄ"
250 PRINT A$+" ﾃﾞｽ｡"
260 RETURN

run
ﾌｧｲﾙ ﾒｲ : ? test
ﾄﾞﾗｲﾌﾞ 1 ﾉ ｿﾞｸｾｲ ﾊ : ｶｲｼﾞｮ ﾃﾞｽ｡
ﾌｧｲﾙ test ﾉ ｿﾞｸｾｲ ﾊ : ｶｲｼﾞｮ ﾃﾞｽ｡
Ok
```

CDBL　[N88] [N]

機　能	整数値，単精度実数値を倍精度実数値に変換した数値を与える.
書　式	CDBL（<数式>）
文　例	A #＝CDBL(B!/2)

解　説　<数式>の値を倍精度実数値に変換します．但し，型変換が行われるだけで有効桁数の変化はありませんから，得られた値の精度は，変換する前の型と同じ（整数型なら整数部のみ，単精度実数型なら有効数字 6 桁）になります．

参照：CINT，CSNG

サンプルプログラム

```
100 REM ｺﾞｻ ﾉ ﾅｲ ﾊﾞｲｾｲﾄﾞｼﾞｯｽｳ ﾍﾉ ﾍﾝｶﾝ
110 A!=1.23456:GOSUB 150:GOSUB 210
120 A!=123456!:GOSUB 150:GOSUB 210
130 A!=1.23456E+08:GOSUB 150:GOSUB 210
140 END
150 A$=STR$(A):A1#=CDBL(A!)
160 D=INSTR(A$,".")
170 IF D=0 OR INSTR(A$,"E")<>0 THEN A2#=A1#:RETURN
180 D=LEN(A$)-D
190 A2#=INT(CDBL(A!*10^D+.5))/10^D
200 RETURN
210 PRINT A!,A1#;TAB(34);A2#
220 RETURN

run
 1.23456        1.234559893608093   1.234558649701477
 123456         123456              123456
 1.23456E+08    123456000           123456000
Ok
```

CHR$　N88 N

機能	指定したキャラクタコードを持つ文字を与える.
書式	CHR$（<数式>）
文例	A$＝CHR$(65)

解説　<数式>の値のキャラクタコードを持つ文字を与えます. 値が 0 ～255の範囲にない場合は, "Illegal function call" (FC Error) エラーが起こります.

参照：ASC

サンプルプログラム

```
100 REM ｵｵﾓｼﾞ -> ｺﾓｼﾞ ﾍﾝｶﾝ
110 INPUT A$
120 IF A$="" THEN 200
130 B$=""
140 FOR I=1 TO LEN(A$)
150  D$=MID$(A$,I,1):D=ASC(D$)
160  IF D<65 OR D>90 THEN B$=B$+D$:GOTO 180
170  B$=B$+CHR$(D+32)
180 NEXT I
190 PRINT:PRINT B$
200 END
```

```
run
? NEC PC-8801MK2 computer

nec pc-8801mk2 computer
OK
```

CINT　[N88] [N]

[機　能]　単精度実数値，倍精度実数値を整数値に変換した値を与える．

[書　式]　CINT（〈数式〉）

[文　例]　A％＝CINT(B＃＊2)

[解　説]　〈数式〉の値の小数点以下を四捨五入（N-BASICでは切り捨て）して整数に変換した値を与えます．結果の値が－32768～32767の範囲にない場合は，〝Overflow″（OV Error）エラーが起こります．

参照：CSNG，CDBL

[サンプルプログラム]

```
100 REM ｼｮｳｽｳﾃﾝｲｶ ﾆｼｲ ﾉ ｹﾀ ﾃﾞﾉ 4ｼｬ5ﾆｭｳ
110 A=1.23456:D=2:GOSUB 140:GOSUB 170
120 A=1.23456:D=3:GOSUB 140:GOSUB 170
130 END
140 IF A=INT(A) THEN AD=A:RETURN
150 AD=CINT(A*10^D)/10^D
160 RETURN
170 PRINT A,AD
180 RETURN

run
 1.23456        1.23
 1.23456        1.235
Ok
```

COS　N88　N

機　能	余弦（コサイン）を与える.
書　式	COS（<数式>）
文　例	C=COS(3.1415／2)

解　説　<数式>の値に対する余弦を与えます.<数式>の単位はラジアンです.

また，<数式>に倍精度実数が含まれる場合は，倍精度の値を与えますが，他の場合には，単精度の値を与えます.

注意：倍精度機能は，N_{88}DISK-BASICモードでのみ有効です.

参照：SIN，TAN

サンプルプログラム

```
100 REM cos ｶﾗ sec(ｾｶﾝﾄ) ｦ ﾓﾄﾒﾙ
110 INPUT "ｶｸﾄﾞ (ﾄﾞ) : ";D
120 CS=COS(D/180*3.1415926#)
130 PRINT "COS";USING "###";D;:PRINT " = ";CS,
140 PRINT "SEC";USING "###";D;
150 IF ABS(CS)<=.000001 THEN 180 ELSE SC=1/CS
160 PRINT " = ";SC
170 END
180 PRINT " = ﾃｲｷﾞ ｻﾚﾏｾﾝ"
190 END

run
ｶｸﾄﾞ (ﾄﾞ) : ? 45
COS 45 =  .707107          SEC 45 =  1.41421
Ok
```

CSNG　N88　N

機　能	整数値，倍精度実数値を単精度実数値に変換した値を与える.
書　式	CSNG（〈数式〉）
文　例	A！＝CSNG(B＃)

解　説　〈数式〉の値を有効数字6桁の単精度実数型に変換します．その値が−1.70141E＋38〜1.70141E＋38の範囲にない場合は，〝Overflow〞（OV Error）エラーが起こります．

参照：CINT，CDBL

サンプルプログラム

```
100 REM ｼｮｳｽｳﾃﾝｲｶ ﾆｼﾝｲ ﾉ ｹﾀ ﾃﾞﾉ 4ｼｬ5ﾆｭｳ
110 A#=1.23456789#:D=1:GOSUB 140:GOSUB 170
120 A#=1.23456789#:D=7:GOSUB 140:GOSUB 170
130 END
140 IF A#=INT(A#) THEN AD#=A#:RETURN
150 AD#=INT(A#*10^D+.5)/10^D
160 RETURN
170 PRINT A#,CSNG(AD#)
180 RETURN

run
 1.23456789     1.2
 1.23456789     1.23457
Ok
```

CSRLIN　N88 N

機　能	現在のカーソルの行位置を与える.
書　式	CSRLIN
文　例	Y＝CSRLIN

解　説　現在のカーソルの行位置を与えます．値の範囲は，25行モードでは0〜24,20行モードでは0〜19となります.なお,画面の最上行が0,最下行が24(または19）となっています（POS関数参照).

参照：POS

サンプルプログラム

```
100 REM ﾋﾞﾘﾔｰﾄﾞ ﾉ ｼﾐｭﾚｰｼｮﾝ
110 LOCATE 1,2:PRINT STRING$(78,"●");
120 LOCATE 1,23:PRINT STRING$(78,"●");
130 FOR I=3 TO 22:LOCATE 0,I:PRINT "●";
140 LOCATE 79,I:PRINT "●";:NEXT
150 LOCATE 0,2:PRINT "●";:LOCATE 79,2:PRINT "●";
160 LOCATE 0,23:PRINT "●";:LOCATE 79,23:PRINT "●";
170 F=1:X=RND(1)*77+1:C=RND(1)*18+3
180 FOR X=X TO 78
190  LOCATE XO,CO:PRINT " ";
200  IF CSRLIN=22 OR CSRLIN=3 THEN F=F*-1
210  C=C+F:LOCATE X,C:PRINT "●";:XO=X:CO=C
220 NEXT X
230 FOR X=78 TO 2 STEP -1
240  LOCATE XO,CO:PRINT " ";
250  IF CSRLIN=22 OR CSRLIN=3 THEN F=F*-1
260  C=C+F:LOCATE X,C:PRINT "●";:XO=X:CO=C
270 NEXT X
280 GOTO 180
```

CVI／CVS／CVD　N88 N　(Nではディスクのみ)

機　能　文字列を数値データに変換した値を与える.

書　式　1) CVI (<2文字の文字列>)
　　　　2) CVS (<4文字の文字列>)
　　　　3) CVD (<8文字の文字列>)

文　例　1) A%=CVI(A$)
　　　　2) B=CVS("A3Bd")
　　　　3) C#=CVD(NUM$)

解　説　ディスケット上のランダムデータ・ファイルから読み込んだデータは，文字型になっているため，CVI，CVS，CVD各関数を使って，数値データに変換しなければなりません．CVIは2文字の文字列（即ち16ビットのデータ）を整数値に，CVSは4文字の文字列を単精度実数値に，CVDは8文字の文字列を倍精度実数値にそれぞれ変換します．

実際の文字列と変換後の数値との関係は，わかりやすく整数型で説明すると次の様になります．

A$="AB"の場合，即ち，A$=CHR$(&H41)+CHR$(&H42)の場合には，CVI(A$)は，&H4241（=16961）になります．つまり，キャラクタコードを数値データとみなすわけです．CVS，CVDの場合は，各キャラクタコードを正規化された浮動小数点形式の数値データとみなすことになります．

参照：MKI$／MKS$／MKD$

サンプルプログラム

```
100 '-- input value from random file --
110 OPEN "cvsmks" AS #1
120 FIELD #1,4 AS A$
130 FOR I=1 TO 3
140   GET #1,I:A=CVS(A$):PRINT A
150 NEXT I
160 END

run
 .245121
 .305003
 .311866
Ok
```

DATE$ N88 N

機　能	日付を与える.
書　式	DATE$
文　例	D$=DATE$

解　説　DATE$には常に現在の日付が入れられており，いつでもその内容を見ることができます.

またDATE$には，YY／MM／DD（"81／12／01" 等）の形で日付をセットできます．他の変数に代入した場合も，同じ形の文字列が代入されます.

注意：PC-8801MKIIはバッテリーバックアップにより常に日付を更新して正しい日付を維持するようになっています。従って不用意に日付を変えると，せっかくの正しい日付がこわされることになります.

また，電源をOFF／ONした場合や，リセットした場合は，年の値だけイニシャライズされ "84" となります.

N_{88}DISK-BASICでは，ディスクユーティリティの「SET UP INFO」で，年の値を設定しておくことができます.

参照：TIME$

サンプルプログラム

```
100 ' "81/12/25" --> "25,Dec.1981"
110  DIM MONTH$(12)
120  FOR I=1 TO 12
130   READ MONTH$(I)
140  NEXT I
150  DATE$="81/12/25"
160  GOSUB *TRNS
170  PRINT DATE$;" --> ";M$
180 END
190 *TRNS
200  M$=RIGHT$(DATE$,2)+","
210  M$=M$+MONTH$(VAL(MID$(DATE$,4,2)))
220  M$=M$+".19"+LEFT$(DATE$,2)
230 RETURN
240  DATA Jan,Feb,Mar,Apr,May,Jun
250  DATA Jul,Aug,Sep,Oct,Nov,Dec

run
81/12/25 --> 25,Dec.1981
Ok
```

DSKF（ディスク）　N88　N

機　能　ディスクの諸元を与える．

書　式　DSKF（〈ドライブ番号〉〔，〈機能〉〕）

文　例　PRINT DSKF(1)

解　説　〈ドライブ番号〉で指定されたディスクに関する情報を関数の値として返します．〈機能〉は値として返す情報の種類を指定します．〈機能〉として指定する値と，その時に返される関数の値の意味は次のようになっています．

省略された時：ディスクの残り容量をクラスタ単位で返す．

0：最大トラック番号
(＝片面当りのトラック数－1）
1：1トラック当りのセクタ数
2：片面ディスク（0を返す）
あるいは両面ディスク（1を返す）
3：1トラック当りのクラスタ数
4：ボリューム当りのクラスタ数
5：ディレクトリトラック番号
6：1クラスタ当りのセクタ数
7：FATの開始セクタ番号
8：FATの終了セクタ番号
9：FATの数
10：ディスク属性の入っているセクタ番号

ドライブの種類やディレクトリ，FAT等については，ユーザース・マニュアルを参照してください．

N-BASICでは，〈機能〉を指定することはできず，返される情報は指定したドライブの残りクラスタ数のみです．なお，N-BASICでは1クラスタは8セクタです．

サンプルプログラム

```
100 INPUT "Drive number";DR
110 PRINT "Track  =";DSKF(DR,0);"/Drive"
120 PRINT "Sector =";DSKF(DR,1);"/Track"
130 PRINT "Disk space =";DSKF(DR)

run
Drive number? 1
Track  = 34 /Drive
Sector = 16 /Track
Disk space = 26
Ok
```

DSKI$（ディスク）　N88　N

機　能	ディスクからの直接読み出しをする．
書　式	1) DSKI$（〈ドライブ番号〉，〈トラック番号〉，〈セクタ番号〉）
	2) DSKI$（〈ドライブ番号〉，〈サーフェス番号〉，〈トラック番号〉，〈セクタ番号〉）
文　例	1) D$=DSKI$（2，19，1）
	2) D$=DSKI$（2，1，19，1）

解　説　通常のファイル操作（OPEN，CLOSEなど）とは無関係に，ディスク上の指定したセクタから直接読み出しを行います．

DSKI$は，そのパラメータにより指定されたセクタ上に書かれているデータ256バイトを0番バッファに読み出すとともに，最初の256文字の文字列を関数の値として返します．

文字列の長さは255文字までしか許されないために，関数の値として256バイトのデータ全てを得ることはできません．そこで，通常のランダムアクセスの場合と同じようにFIELD文により，0番バッファへ複数の文字列変数を割り当てることによりこれを解決することができます．あるいは，VARPTR関数により，0番バッファの置かれているメモリ番地を調べた上でPEEK文によりバッファからデータを読み出すこともできます．

〈サーフェス番号〉は，指定された〈ドライブ番号〉のディスクが両面ディスクの場合にのみ必要で，0または1で指定します。

〈トラック番号〉，〈セクタ番号〉として指定できる値の範囲は，指定された〈ドライブ番号〉のディスクドライブの種類によって異なりますが，BASICは，これらの値の範囲を自動的に調べ，不当であった場合には"Illegal function call"（FC Error）エラーとします．詳しくはユーザース・マニュアルを参照して下さい．

DSKI$は対象とするディスクドライブが片面ディスクであるか両面ディスクであるかによってダイナミックに書式が変ります．

1)の書式は片面ディスクを対象とする場合であり，〈サーフェス番号〉の指定は必要ありません。

2)の書式は両面ディスクを対象とする場合であり，〈サーフェス番号〉を指定しなければなりません。

注意：複数のディスクが接続されている場合，それぞれのディスクの諸元については DSKF により調べることができます.

参照：DSKO$, FIELD, DSKF, VARPTR

サンプルプログラム

```
100 '--- Read from disk (direct) ---
110  FIELD #0,128 AS A$(0),128 AS A$(1)
120 *START:INPUT "Drive";DR
130  IF DR<1 OR DR>8 THEN *START
140  IF DSKF(DR,2)=0 THEN *ENTER
150  INPUT "Surface";SU
160  IF SU<0 OR SU>1 THEN 150
170 *ENTER:INPUT "Track,Sector";TR,SE
180  IF TR<0 OR TR>DSKF(DR,0) THEN *ENTER
190  IF SE<1 OR SE>DSKF(DR,1) THEN *ENTER
200  IF DSKF(DR,2)=0 THEN *SINGLE
210  DUMMY$=DSKI$(DR,SU,TR,SE):GOTO *DISP
220 *SINGLE:DUMMY$=DSKI$(DR,TR,SE)
230 *DISP:FOR I=0 TO 1
240   FOR J=0 TO 7
250    FOR K=1 TO 16
260     D$=HEX$(ASC(MID$(A$(I),J*16+K,1)))
270     PRINT RIGHT$("0"+D$,2);" ";
280    NEXT K:PRINT
290  NEXT J,I:END

run
Drive? 1
Track,Sector? 18,1
73 63 72 65 65 6E 31 20 20 00 23 FF FF FF FF FF
69 6E 69 20 20 20 20 20 20 00 22 FF FF FF FF FF
6B 61 6E 6A 69 63 6F 64 20 00 26 FF FF FF FF FF
74 65 73 74 20 20 20 20 20 00 21 FF FF FF FF FF
72 61 6E 64 6F 6D 69 7A 65 00 27 FF FF FF FF FF
72 65 6D 6F 76 65 20 20 20 00 20 FF FF FF FF FF
72 6F 6C 6C 20 20 20 20 20 00 28 FF FF FF FF FF
73 61 76 65 20 20 20 20 20 00 1F FF FF FF FF FF
73 63 72 65 65 6E 32 20 20 00 29 FF FF FF FF FF
73 65 74 20 20 20 20 20 20 00 1E FF FF FF FF FF
74 65 72 6D 20 20 20 20 20 00 2A FF FF FF FF FF
74 69 6D 65 24 6F 6E 20 20 00 1D FF FF FF FF FF
76 69 65 77 28 73 29 20 20 00 2B FF FF FF FF FF
77 68 69 6C 65 77 65 6E 64 00 1C FF FF FF FF FF
77 69 64 74 68 31 20 20 20 00 2C FF FF FF FF FF
77 69 64 74 68 32 20 20 20 00 1B FF FF FF FF FF
OK
```

EOF　N88 N　（Nではディスクのみ）

機　能	ファイルの終了コードを与える．
書　式	EOF（<ファイル番号>）
文　例	IF EOF(1) THEN CLOSE 1 : END

解　説　<ファイル番号>で指定されたファイルが終りに達したかどうかを調べる関数です．関数の値として終りに達していれば真(－1)，そうでなければ偽（0）を返します．<ファイル番号>で指定されたファイルは入力モードでオープンされていなければなりません．

指定されたファイルがRS－232Cコミュニケーション・チャネル（COM：）であった場合には，EOF関数は，バッファが空であった場合に値を真とします．

E

注意：カセットを使用している場合は，EOF関数を使うことはできません．

サンプルプログラム

```
100 '-- data input --
110 OPEN "2:data5" FOR INPUT AS #1
120 IF EOF(1) THEN 160
130  INPUT #1,A
140  PRINT A
150 GOTO 120
160 CLOSE

run
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
Ok
```

ERL／ERR　N88　N

機　能　発生したエラーのエラーコード及びエラーの発生した行番号を与える.

書　式　ERL
ERR

文　例　L＝ERL
E＝ERR

解　説　エラーが発生した時点で，システム変数ERRはエラーコードを，ERLはエラーの発生した行番号を持っています．エラーがダイレクトモードの実行によって生じた場合，ERLは行番号として65535を持ちます.

一般に，ERR及びERLはON ERROR GOTO文によって指定したエラー処理ルーチンにおいて，処理の流れを制御するために使われます.

参照：ON ERROR GOTO

サンプルプログラム

```
100 '-- X ﾉ Y ｼﾞｮｳｦ ﾓﾄﾒﾙ｡ --
110  ON ERROR GOTO *ERRORTRP
120 *START
130  INPUT "X,Y";X,Y
140  ANS=X^Y
150  PRINT X;"ﾉ";Y;"ｼﾞｮｳ ﾊ";ANS;"ﾃﾞｽ｡"
160  GOTO *START
170  PRINT " ??? ﾃｲｷﾞ ｻﾚﾏｾﾝ"
180  END
190 '-- error routine --
200 *ERRORTRP
210  IF ERR=11 AND ERL=140 THEN RESUME 170
220  ON ERROR GOTO 0

run
X,Y? 2,3
 2 ﾉ 3 ｼﾞｮｳ ﾊ 8 ﾃﾞｽ｡
X,Y? 10,4
 10 ﾉ 4 ｼﾞｮｳ ﾊ 10000 ﾃﾞｽ｡
X,Y? -1,3
-1 ﾉ 3 ｼﾞｮｳ ﾊ-1 ﾃﾞｽ｡
X,Y? 0,20
 0 ﾉ 20 ｼﾞｮｳ ﾊ 0 ﾃﾞｽ｡
X,Y? 0,-1
 ??? ﾃｲｷﾞ ｻﾚﾏｾﾝ
Ok
```

EXP　N88 N

機　能　eに対する指数関数の値を与える．

書　式　EXP（〈数式〉）

文　例　E＝EXP(1)

解　説　〈数式〉の指数関数の結果を値とします．但し，〈数式〉の値が87.33655より大きい場合には〝Overflow〞（OV Error）エラーが起こります（N_{88}-BASIC単精度の場合）．

また，〈数式〉に倍精度実数が含まれる場合は倍精度の値を与えますが，他の場合には単精度の値を与えます．

サンプルプログラム

```
100 '    EXP ｶﾗ SINH(ﾊｲﾊﾟﾌﾞﾘｯｸ ｻｲﾝ)
110 '           COSH(ﾊｲﾊﾟﾌﾞﾘｯｸ ｺｻｲﾝ) ｦ ﾓﾄﾒﾙ
120 INPUT D
130 HS=(EXP(D)-EXP(-D))/2:HC=(EXP(D)+EXP(-D))/2
140 PRINT "SINH";USING "###";D;:PRINT " = ";HS,
150 PRINT "COSH";USING "###";D;:PRINT " = ";HC,
160 END

run
? 2
SINH  2 =  3.62686          COSH  2 =  3.7622
Ok
```

FIX　N88　N

機　能	整数部を与える．
書　式	FIX（〈数式〉）
文　例	F=FIX(B／3)

解　説　〈数式〉の値の小数点以下を取り去った値を与えます．
INT との違いはサンプルプログラムを見れば明らかです．

参照：INT

サンプルプログラム

```
100 '--  fix & int  --
110 A=-2
120 FOR I=0 TO 9
130 A=A+I*.3
140 PRINT "A =";A,"FIX(A) =";FIX(A),"INT(A) =";INT(A)
150 NEXT I

run
A =-2          FIX(A) =-2     INT(A) =-2
A =-1.7        FIX(A) =-1     INT(A) =-2
A =-1.1        FIX(A) =-1     INT(A) =-2
A =-.2         FIX(A) = 0     INT(A) =-1
A = 1          FIX(A) = 1     INT(A) = 1
A = 2.5        FIX(A) = 2     INT(A) = 2
A = 4.3        FIX(A) = 4     INT(A) = 4
A = 6.4        FIX(A) = 6     INT(A) = 6
A = 8.8        FIX(A) = 8     INT(A) = 8
A = 11.5       FIX(A) = 11    INT(A) = 11
Ok
```

F

FPOS　N88　N　（Nではディスクのみ）

機　能	ファイル中での物理的な現在位置を与える.
書　式	FPOS（〈ファイル番号〉）
文　例	HEAD＝FPOS(4)

解　説　〈ファイル番号〉によって指定されたファイルが最後に読み書きした位置を返します.

指定されたファイルがディスクファイルの場合には，最後に読み書きしたセクタの番号を返します．この番号はトラック0，サーフェス0，セクタ1を0番とした通し番号で与えられます.

指定されたファイルがプリンタであった場合には，プリンタのヘッドの位置を与えます．この値はLPOSが返す値と同一です.

N-BASICでは，FPOSはディスクファイルに対してのみ用いることができます.

参照：LOC，LPOS

サンプルプログラム

```
100 '-- fpos value print out --
110 OPEN "2:data6" FOR OUTPUT AS #1
120 PRINT FPOS(1):PRINT
130 FOR I=1 TO 5
140  PRINT #1,STRING$(128,"1")
150  PRINT #1,STRING$(128,"1")
160  PRINT FPOS(1)
170 NEXT I
180 CLOSE

run
 192

 194
 195
 196
 197
 198
Ok
```

FRE　N88　N

機　能	メモリの未使用領域の大きさを与える.
書　式	FRE (<機能>)
文　例	PRINT FRE(0)

解　説　FREは, BASICの利用可能なメモリ領域のうちの, 未使用領域の大きさを関数値として返します. <機能>は, 0, 1, 2の値をとり, それぞれの場合に返される関数値の意味は次のとおりです.

0 : 未使用変数領域のバイト数. 単純変数, 配列及びストリングの領域として使用可能な領域の残りバイト数です.

1 : 未使用テキスト領域のバイト数. プログラムテキストを入れるための領域の残りバイト数です.

2 : 未使用変数領域と未使用テキスト領域の和のバイト数.

<機能>が0または2の時は, FRE関数は, ストリング領域の整理のための〝ちり集め〟処理を必ず行いますので, 関数が値を返すまでに時間がかかることがあります.

N - BASICでは, <機能>が数値であればメモリの未使用領域の大きさ, 文字であれば未使用の文字領域の大きさをバイト数で返します.

N88 - BASICでは, <機能> が文字列の場合には, 〝Illegal function call〟 (FC Error) エラーとなります.

サンプルプログラム

```
100 '-- User Memory print out --
110 DATA Valiable memory,Text memory,Total
120 FOR I%=0 TO 2
130   READ A$:FR=FRE(I%)
140   PRINT A$;TAB(18);FR;TAB(26);HEX$(FR);"H"
150 NEXT I%

run
Valiable memory    13247   33BFH
Text memory        32581   7F45H
Total              45838   B30EH
Ok
```

HEX$ N88 N

機　能	10進数を16進数に変換し，その文字列を与える．
書　式	HEX$ (〈数式〉)
文　例	A$＝HEX$(X)

解　説　〈数式〉の値を16進数に変換して，その文字列を与える関数です．〈数式〉の値は，－32768から32767（または0から65535）までの範囲になければなりません．

参照：VAL，OCT$

サンプルプログラム

```
100 PRINT "     -0 -1 -2 -3 -4 -5 -6 -7 -8 -9"
110 FOR I=0 TO 9
120 PRINT STR$(I);"- ";
130  FOR J=0 TO 9
140   PRINT RIGHT$("0"+HEX$(I*10+J),2);" ";
150  NEXT J
160  PRINT
170 NEXT I
180 END

run
     -0 -1 -2 -3 -4 -5 -6 -7 -8 -9
 0- 00 01 02 03 04 05 06 07 08 09
 1- 0A 0B 0C 0D 0E 0F 10 11 12 13
 2- 14 15 16 17 18 19 1A 1B 1C 1D
 3- 1E 1F 20 21 22 23 24 25 26 27
 4- 28 29 2A 2B 2C 2D 2E 2F 30 31
 5- 32 33 34 35 36 37 38 39 3A 3B
 6- 3C 3D 3E 3F 40 41 42 43 44 45
 7- 46 47 48 49 4A 4B 4C 4D 4E 4F
 8- 50 51 52 53 54 55 56 57 58 59
 9- 5A 5B 5C 5D 5E 5F 60 61 62 63
Ok
```

H

INKEY$ N88 N

機　能	キーが押されていれば，その文字を，キーが押されなければヌルストリングを与える.
書　式	INKEY$
文　例	A$=INKEY$

解　説　キーボードバッファが空であれば，INKEY$関数はヌルストリングを返します．キー入力によりキーボードバッファが空でない場合には，バッファの先頭から1文字取り出し，その文字を返します．

なお，CTRL - C，ストップキーはINKEY$関数によって拾うことはできません．また押されたキーは画面上には表示されません．

サンプルプログラム

```
100  PRINT "Enter any key to end ··"
110 *LOOP
120  A$=INKEY$
130   IF A$="" THEN *LOOP
140   PRINT A$
150   IF A$="e" THEN *EXIT ELSE *LOOP
160 *EXIT:END

run
Enter any key tq end ··
n
8
8
-
b
a
s
i
c

e
Ok
```

I

INP　N88　N

機　能　入力ポートから値を得る.

書　式　INP (〈ポート番号〉)

文　例　A=INP(15)

解　説　〈ポート番号〉で指定された入力ポートから8ビットのデータを読み取り，それを関数値とします．〈ポート番号〉として指定できる値の範囲は0から255までです．

参照：OUT

サンプルプログラム

```
100  PRINT "Type 'e' key to end .."
110 *LOOP
120  IF INP(2)=223 THEN END
130 '  We  have to  dummy read
140 ' to clear the key buffer.
150  DUM$=INPUT$(1):PRINT DUM$
160 GOTO *LOOP

run
Type 'e' key to end ..
a
b
c
d
e
Ok
```

INPUT$ N88 N （Nではディスクのみ）

機　能	指定されたファイルより指定された長さの文字を与える．
書　式	INPUT$ (<文字数>〔，〔#〕<ファイル番号>〕)
文　例	WORD$=INPUT$(6, #2)

解　説　〈ファイル番号〉によって指定されたファイルから〈文字数〉分の文字列を読み出します．〈ファイル番号〉が省略された場合には，キーボードから入力が行われますが，INPUT 文と異なり入力された文字はスクリーンには表示されません．

INPUT$は指定された〈文字数〉の文字が入力されるのを待ち続けますが，既にバッファに入力済みのデータがある場合にはバッファ中から文字を拾ってきます．また INPUT$は，ストップキー，CTRL-C を除くすべての文字をそのまま読み出しますので，INPUT 文や LINE INPUT 文では入力することのできない改行文字等も入力することができます．

但し，N - BASIC ではディスク版でのみ〈ファイル番号〉の指定ができます．

参照：INPUT，LINE INPUT

サンプルプログラム

```
100 '-- get key --
110 FOR I=1 TO 5
120  A$=INPUT$(I)
130  PRINT A$
140 NEXT I
150 END

run
a
bc
def
ghij
klmno
Ok
```

INPUT$ [N]

機　能	**バッファより指定された長さの文字を与えます。**
書　式	INPUT $(〈文字数〉〔, %〈ポート番号〉〕)
文　例	A$＝INPUT$(6, %1)

解　説　〈文字数〉は０～255の範囲でなければなりません．

〈ポート番号〉を指定するとRS-232Cのバッファより〈文字数〉分の文字列を読み出します．使い方は，ファイルから文字を読み出すINPUT$と同じです．

サンプルプログラム

```
100 '-- RS-232C port input --
110 'switch to RS-232C channel
120 OUT &H30,&H20 OR (&HF AND PEEK(&HEA66))
130 POKE &H8001,&H23:POKE &H8000,&H7F
140 OUT &HE6,4
150 OUT &HE4,&HFF
160 INIT %1,&HFA,&H37
170 IF PORT(1) THEN PRINT INPUT$(PORT(1),%1);
180 C$=INKEY$:IF C$<>"" THEN PRINT %1,C$;
190 GOTO 170
200 '*******************************
210 ' ﾌﾟﾛｸﾞﾗﾑ ｼｭｳﾘｮｳ ｺﾞ out &hE6,&h0
220 ' ｦ ｼﾞｯｺｳ ｼﾃ ｸﾀﾞｻｲ｡
230 '*******************************
```

INSTR　N88　N

機能　**文字列の中から任意の文字列を捜してその文字の位置を与える．**

書式　INSTR（〔〈数式〉,〕〈文字列 1〉,〈文字列 2〉）

文例　J＝INSTR(A$,"J")

解説　〈文字列 1〉の中から〈文字列 2〉を探し，みつかれば，発見した位置を，見つからなければ 0 を値として返します．

〈数式〉は捜し始める位置を指定します．もし省略した場合は〈文字列 1〉の最初から捜し始めます．

〈文字列 2〉にヌルストリングを指定すると，〈数式〉と同じ値を返します．

サンプルプログラム

```
100 '-- alphabet search --
110  A$="abcdefghijklmnopqrstuvwxyz"
120  ALP$=INPUT$(1)
130  ALP=ASC(ALP$)
140   IF ALP>64 AND ALP<91 THEN ALP=ALP+32
150  A=INSTR(A$,CHR$(ALP))
160   IF A=0 THEN PRINT ALP$;" ﾊ ｱﾙﾌｧﾍﾞｯﾄ ﾃﾞﾊｱﾘﾏｾﾝ":END
170  PRINT USING "! ﾊ ## ﾊﾞﾝﾒﾉ ｱﾙﾌｧﾍﾞｯﾄﾃﾞｽ";ALP$,A
180 END

run
F ﾊ  6 ﾊﾞﾝﾒﾉ ｱﾙﾌｧﾍﾞｯﾄﾃﾞｽ
OK
```

INT　N88　N

機　能	小数点以下を切り捨てた整数値を与える.
書　式	INT (<数式>)
文　例	PRINT INT (1.123)

解　説　<数式> の値を超えない最大の整数を与えます.

参照：FIX, CINT

サンプルプログラム

```
100 ' ｾｲｽｳ ﾄ ｼﾞｯｽｳ ﾉ ｸﾍﾞﾂ
110  A$(0)="ｼﾞｯｽｳ ﾃﾞｽ"
120  A$(1)="ｾｲｽｳ ﾃﾞｽ"
130  FLG=0
140  INPUT A
150   IF A=INT(A) THEN FLG=1
160  PRINT A;" ﾊ ";A$(FLG)
170 END

run
? 123
 123  ﾊ ｾｲｽｳ ﾃﾞｽ
Ok
run
? 87.4657
 87.4657  ﾊ ｼﾞｯｽｳ ﾃﾞｽ
Ok
```

LEFT$　N88 N

機　能　文字列の左側から任意の長さの文字列を与える.

書　式　LEFT$ (〈文字列〉, 〈数式〉)

文　例　B$=LEFT$(A$,4)

解　説　〈数式〉は0から255の範囲になければならない.〈数式〉が,〈文字列〉の総文字数以上の場合は,〈文字列〉のすべてを結果とします.〈数式〉が0ならば,ヌルストリングを結果とします.

サンプルプログラム

```
100 ' モジレツ(10) ノ カイテン ( <- )
110  INPUT A$
120  FOR I=1 TO 10
130   PRINT A$
140   B$=LEFT$(A$,1)
150   A$=MID$(A$,2)+B$
160  NEXT I
170 END

run
? 1234567890
1234567890
2345678901
3456789012
4567890123
5678901234
6789012345
7890123456
8901234567
9012345678
0123456789
Ok
```

L

LEN　Ⓝ88 Ⓝ

機　能	文字列の総文字数を与える.
書　式	LEN (<文字列>)
文　例	PRINT LEN(A$)

解　説　出力されない文字や空白も数えます．出力されない文字とはキャラクタコードで１から31までの通常コントロールコードと呼ばれるものをいいます．

サンプルプログラム

```
100   INPUT A$
110   L=LEN(A$)
120   GOSUB *FLAME
130   PRINT "* ";A$;" *"
140   GOSUB *FLAME
150  END
160  *FLAME
170   FOR I=1 TO L+4
180    PRINT "*";
190   NEXT I
200   PRINT
210  RETURN
```

```
run
? NEC PC-8801MK2 N88-basic
****************************
* NEC PC-8801MK2 N88-basic *
****************************
Ok
```

LOC　N88　N　（Nではディスクのみ）

機能　ファイル中での論理的な現在位置を与える．

書式　LOC（<ファイル番号>）

文例　LAST＝LOC（2）

解説　<ファイル番号>で指定されたファイルがランダムファイルであった場合には，LOC関数は，そのランダムファイルに対して最後に読み書き（GET/PUT）されたレコード番号を返します．

<ファイル番号>で指定されたファイルが，シーケンシャルファイルであった場合には，そのファイルがオープンされてから，読み書きされたレコード数を返します．

指定されたファイルがRS-232Cコミュニケーション・ファイルであった場合には，LOC関数は，割り込みによって受けつけられて，入力バッファ中にたまっている文字数を値として返します．読み取り動作（INPUT #，INPUT$等）が行われずに，割り込みによるRS-232Cチャネルからの受信が続くと，やがてバッファが一杯になりエラーとなります．

指定されたファイルがキーボードファイルであった場合には，LOC関数はキーボードからの割り込みによって受けつけられて，キーボードバッファにたまっている文字数を返します．

N-BASICでは，LOC関数は，ディスクファイルに対してのみ用いることができます．

サンプルプログラム

```
100 OPEN "2:data6" AS #1
110 FIELD #1,20 AS A$,8 AS B$
120 INPUT "ﾅﾏｴ";NA$:INPUT "ｷｭｳﾖ";KY#
130 LSET A$=NA$:RSET B$=MKD$(KY#):PUT 1
140 PRINT NA$;" ｻﾝﾉ data ﾊ";LOC(1)"ﾊﾞﾝﾃﾞｽ｡"
150 INPUT "continue?(y/n)";C$
160 IF C$="y" THEN PRINT:GOTO 120
170 IF C$<>"n" THEN 150 ELSE END
run
ﾅﾏｴ? ﾑﾗﾏﾂ
ｷｭｳﾖ? 150000
ﾑﾗﾏﾂ ｻﾝﾉ data ﾊ 1 ﾊﾞﾝﾃﾞｽ｡
continue?(y/n)? n
OK
```

LOF　[N88] [N]　（[N]ではディスクのみ）

機　能	ファイルの大きさを与える.
書　式	LOF (〈ファイル番号〉)
文　例	MAXREC＝LOF(2)

解　説　〈ファイル番号〉により指定されたファイルがディスクファイルである場合には, LOC 関数はそのフィルムの大きさをセクタ数で返します. この値は, ファイルがラムダムファイルであれば, そのファイルの最大レコード番号に相当します.

指定されたファイルが RS - 232C コミュニケーション・ファイルである場合には, LOF 関数は, バッファの残りバイト数を返します.

N - BASIC では, LOF 関数は, ディスクファイルに対してのみ用いることができます.

サンプルプログラム

```
100 '-- LOF of sequntial & random file --
110 OPEN "2:sdata1" FOR OUTPUT AS #1
120 PRINT #1,123:CLOSE
130 OPEN "2:sdata1" FOR INPUT AS #1
140 OPEN "2:rdata1" AS #2
150 FIELD #2,128 AS A$,128 AS B$
160 PRINT "sdata1 ﾉ ｾｸﾀ ｽｳﾊ";LOF(1);"ﾃﾞｽ｡"
170 PRINT "rdata1 ﾉ ﾚｺｰﾄﾞｽｳﾊ";LOF(2);"ﾃﾞｽ｡"
180 CLOSE

run
sdata1 ﾉ ｾｸﾀ  ｽｳﾊ 1 ﾃﾞｽ｡
rdata1 ﾉ ﾚｺｰﾄﾞｽｳﾊ 0 ﾃﾞｽ｡
OK
```

LOG　N88　N

機　能	自然対数を与える.
書　式	LOG（〈数式〉）
文　例	PRINT LOG(35/9)

解　説　〈数式〉によって与えられた値の自然対数（e を底とした対数）を返します.

　また，〈数式〉に倍精度実数が含まれる場合は，倍精度の値を与えますが，他の場合には，単精度の値を与えます.

　対照：EXP

サンプルプログラム

```
100 ' ｼﾞｮｳﾖｳ ﾀｲｽｳ ｦ ﾓﾄﾒﾙ
110  F$="log10(##.#####^^^^)=##.#####^^^^"
120  INPUT L
130  A=LOG(10)
140  B=LOG(L)
150  PRINT USING F$;L,B/A
160 END

run
? 100
log10( 1.00000E+02)= 2.00000E+00
Ok
run
? .1245
log10( 1.24500E-01)=-9.04831E-01
Ok
run
? 876.2
log10( 8.76200E+02)= 2.94260E+00
Ok
```

LPOS　N88 N

機能	現在のプリンタのヘッド位置を与えます.
書式	LPOS（<式>）
文例	LPOS(0)

解説　<式> の値はダミーであり意味を持ちません.

参照：WIDTH LPRINT

サンプルプログラム

```
100 WIDTH "lpt1:",255
110 FOR I=0 TO 20
120  IF LPOS(0)>=40 THEN LPRINT
130  LPRINT SQR(I),
140 NEXT I
150 END

run
 0              1              1.41421
 1.73205        2              2.23607
 2.44949        2.64575        2.82843
 3              3.16228        3.31662
 3.4641         3.60555        3.74166
 3.87298        4              4.12311
 4.24264        4.3589         4.47214
Ok
```

MAP　N88

機　能	**スクリーン座標，ワールド座標の相互変換を行う．**
書　式	MAP（〈数式1〉，〈機能〉）
文　例	SY1＝MAP(WY1，1)

解　説　〈数式〉で指定されるスクリーン座標あるいはワールド座標を，〈機能〉の指定に従い，対応するワールド座標あるいはスクリーン座標に変換します．〈機能〉は変換の種類を指定し，0から3までの値をとり，それぞれ次の意味を持ちます．

0：〈数式〉をワールド座標系における x 座標とみなし，これを対応するスクリーン座標系での x 座標に変換します．(Wx → Sx)

1：〈数式〉をワールド座標系における y 座標とみなし，これを対応するスクリーン座標系での y 座標に変換します．(Wy → Sy)

2：〈数式〉をスクリーン座標系における x 座標とみなし，これを対応するワールド座標系での x 座標に変換します．(Sx → Wx)

3：〈数式〉をスクリーン座標系における y 座標とみなし，これを対応するワールド座標系での y 座標に変換します．(Sy → Wy)

MAP関数を用いれば，ワールド座標からスクリーン座標の変換ができますから，GET文やPUT文のように，スクリーン座標で位置を指定する操作の場合にもワールド座標を変換して指定することができます．

注意：MAP関数は，変換の結果がスクリーン座標系あるいはワールド座標系から外れてもエラーにならないので注意してください．

参照：1章・ウィンドウとビューポート

サンプルプログラム

```
100 '** World <--> Screen **
110  SCREEN 0,0
120  WINDOW(-100,-100)-(100,100)
130  PRINT TAB(5);"window(-100,-100)-(100,100)"
140  PRINT TAB(5);"view(0,0)-(639,199)"
150  PRINT "ニ オイテ World <--> Screen ノ ヘンカンヲ シマス。"
160  PRINT
170 *ENTRY
180  PRINT "World --> Screen : 1"
190  PRINT "Screen --> World : 2"
200  INPUT ANS
210  IF ANS<>1 AND ANS<>2 THEN *ENTRY
220  INPUT "Coordinate =";CO
230  IF ANS=2 THEN *WFROMS
240  PRINT "WX :";CO,"SX =";MAP(CO,0)
250  PRINT "WY :";CO,"SY =";MAP(CO,1)
260  GOTO *ENTRY
270 *WFROMS
280  PRINT "SX :";CO,"WX =";MAP(CO,2)
290  PRINT "SY :";CO,"WY =";MAP(CO,3)
300  GOTO *ENTRY
```

```
run
    window(-100,-100)-(100,100)
    view(0,0)-(639,199)
ニ オイテ World <--> Screen ノ ヘンカンヲ シマス。

World --> Screen : 1
Screen --> World : 2
? 1
Coordinate =? 0
WX : 0          SX = 320
WY : 0          SY = 100
World --> Screen : 1
Screen --> World : 2
? 2
Coordinate =? 100
SX : 100        WX =-68.7011
SY : 100        WY = .50251
World --> Screen : 1
Screen --> World : 2
?
Break in 200
Ok
```

MID$　N88　N

機　能　文字列の中から任意の長さの文字列を与える．

書　式　MID$（〈文字列〉，〈式1〉〔，〈式2〉〕）

文　例　PRINT MID$（A$, 2, 3）

解　説　〈式2〉は0から255の範囲，また，〈式1〉は1から255の範囲にならなければなりません．

また，〈式2〉を省略した場合や，〈文字列〉の〈式1〉番目の文字から右の文字数が〈式2〉より小さい場合は，〈文字列〉の〈式1〉番目より右すべての文字列を結果とします．

〈文字列〉の文字数が，〈式1〉より小さければ，MID$関数は，ヌルストリングを与えます．

参照：RIGHT$，LEFT$

サンプルプログラム

```
100 '-- break of strings --
110  INPUT A$
120  PRINT A$;" = ";
130  FOR I=1 TO LEN(A$) STEP 2
140   PRINT MID$(A$,I,2);" + ";
150  NEXT
160  PRINT CHR$(8);CHR$(8);" "
170 END
run
? 1234567890abcdef
1234567890abcdef = 12 + 34 + 56 + 78 + 90 + ab + cd + ef
Ok
```

MKI$／MKS$／MKD$　　N88 N （Nではディスクのみ）

機能　各数値を内部表現に対応した文字コードに変換する．

書式　1) MKI$（<整数表記>）
2) MKS$（<単精度表記>）
3) MKD$（<倍精度表記>）

文例　1) A$＝MKI$(128)
2) LSET B$＝MKS$(1.23)
3) RSET C$＝MKD$(3.141592654#)

解説　これらの関数は，数値をランダムファイル・バッファに対してLSET／RSET命令で書き込む際に使用します．数値から文字への変換は数値が持つ内部表現（2進数表現）の値をそのままそれに対応する文字コードにすることによって行われます．この逆の動作をする関数としてはCVI／CVS／CVDが用意されています．

1）整数値を2文字（2バイト）の文字列に変換します．
2）単精度数値を4文字（4バイト）の文字列に変換します．
3）倍精度数値を8文字（8バイト）の文字列に変換します．

注意：STR$関数とは変換の仕方が異なります．

参照：CVI, CVS, CVD, STR$

サンプルプログラム

```
100 '-- random data put --
110 OPEN "2:rdata" AS #1
120 FIELD #1,2 AS D1$,4 AS D2$,8 AS D3$
130 A%=123
140 B=8534.13
150 C#=3.14159265400000 2#
160 LSET D1$=MKI$(A%)
170 LSET D2$=MKS$(B)
180 LSET D3$=MKD$(C#)
190 PUT #1,1
200 '-- random data get --
210 GET #1,1
220 A%=CVI(D1$)
230 B=CVS(D2$)
240 C#=CVD(D3$)
250 PRINT "A =";A%
```

```
260 PRINT "B =";B
270 PRINT "C =";C#
280 CLOSE
290 END

run
A = 123
B = 8534.13
C = 3.141592654000002
Ok
```

OCT$　N88 N

機　能	10進数を8進数に変換し，その文字列を与える.
書　式	OCT$ (〈数式〉)
文　例	PRINT OCT$(A)

解　説　〈数式〉の値の範囲は－32768から32767（または0から65535）までです.

参照：HEX$

サンプルプログラム

```
100 '** decimal --> octal --> hexadecimal **
110 FOR I=0 TO 16
120   DE$=RIGHT$("   "+STR$(I),3)
130   OC$=RIGHT$("   "+OCT$(I),3)
140   HE$=RIGHT$("   "+HEX$(I),3)
150  PRINT "DECIMAL:";DE$,"OCTAL:";OC$,"HEX:";HE$
160 NEXT I

run
DECIMAL:  0    OCTAL:  0     HEX:  0
DECIMAL:  1    OCTAL:  1     HEX:  1
DECIMAL:  2    OCTAL:  2     HEX:  2
DECIMAL:  3    OCTAL:  3     HEX:  3
DECIMAL:  4    OCTAL:  4     HEX:  4
DECIMAL:  5    OCTAL:  5     HEX:  5
DECIMAL:  6    OCTAL:  6     HEX:  6
DECIMAL:  7    OCTAL:  7     HEX:  7
DECIMAL:  8    OCTAL: 10     HEX:  8
DECIMAL:  9    OCTAL: 11     HEX:  9
DECIMAL: 10    OCTAL: 12     HEX:  A
DECIMAL: 11    OCTAL: 13     HEX:  B
DECIMAL: 12    OCTAL: 14     HEX:  C
DECIMAL: 13    OCTAL: 15     HEX:  D
DECIMAL: 14    OCTAL: 16     HEX:  E
DECIMAL: 15    OCTAL: 17     HEX:  F
DECIMAL: 16    OCTAL: 20     HEX: 10
Ok
```

PEEK　N88 N

機　能　メモリ上の指定された番地の内容を読み出す.

書　式　PEEK (〈番地〉)

文　例　A=PEEK(&HA000)

解　説　〈番地〉によって指定されたメモリ番地の内容を与えます. 〈番地〉は, 0～65535 (&H0～&HFFFF) の値でなければならず, もし小数点以下がある場合, 整数化してから実行します.

与えられる数値は1バイトの整数です.

但し, PC-8801MKIIのメモリはバンク構成になっているため, N_{88}, N-BASICで読み出すメモリが異なります. N_{88}-BASICではメインバンクの変数エリアとN_{88}-BASIC ROMエリアであり, N-BASICでは変数エリアとN-BASIC ROMエリアです.

参照：POKE

サンプルプログラム

```
100 '-- memory dump --
110  INPUT "start address(hex)";SA$
120  SA=VAL("&h"+SA$)
130  INPUT "end   address(hex)";EA$
140  EA=VAL("&h"+EA$)
150  CO=1
160 *START
170  PRINT RIGHT$("000"+HEX$(SA),4);": ";
180  FOR I=SA TO SA+15
190   DA=PEEK(I)
200   PRINT RIGHT$("0"+HEX$(DA),2);" ";
210  NEXT I
220  SA=SA+16:PRINT
230  IF SA<=EA THEN *START
240 END

run
start address(hex)? 8000
end   address(hex)? 8050
8000: 00 1A 00 64 00 3A 8F E8 2D 2D 20 6D 65 6D 6F 72
8010: 79 20 64 75 6D 70 20 2D 2D 00 3A 00 6E 00 20 85
8020: 20 22 73 74 61 72 74 20 61 64 64 72 65 73 73 28
8030: 68 65 78 29 22 3B 53 41 24 00 4F 00 78 00 20 53
8040: 41 EF FF 94 28 22 26 68 22 F1 53 41 24 29 00 6F
8050: 00 82 00 20 85 20 22 65 6E 64 20 20 20 61 64 64
Ok
```

PEN　N88

機　能	ライトペンからの情報を与える.
書　式	PEN (<機能>)
文　例	X=PEN(1)

解　説　ライトペンが，現在どのような状態にあるかの情報を与えます．

ライトペンからの情報には，ライトペンを押した瞬間のもの（トリガセンス）とライトペンが押されている間のもの（レベルセンス）があり，<機能>の値によって次のように異なります．

PEN(0)：トリガセンス

ペンを押したら真（－1）を，ペンが押されていないと偽（0）を与えます．この関数は，一度真になるとON PEN GOSUB文で定義されている割り込み処理ルーチンに入るまでその値を保持し，処理ルーチンに入った時偽となります．

PEN(1)：レベルセンス

ペンが押されている限り，現在ペンが動作しているキャラクタ座標の水平座標を与えます．

PEN(2)：レベルセンス

ペンが押されている限り，現在ペンが動作しているキャラクタ座標の垂直座標を与えます．

注意：<式>の値はここで示す0～2までで，それ以外はエラーとなります．
またこの関数はPEN ONの状態でなければ値を返しません．

参照：ON PEN GOSUB，PEN ON/OFF/STOP

サンプルプログラム

```
100 SCREEN 1,0:CONSOLE ,,,1
110 COLOR 0,7:CLS 2
120 PEN ON:ON PEN GOSUB *PENSUB
130 GOTO 130
140 *PENSUB:PRINT "X:";PEN(1),"Y:";PEN(2)
150 RETURN
run
X: 26          Y: 3
X: 55          Y: 14
```

［1］POINT　N88

機　能	Last Referenced Point（LP）の値を与える.
書　式	POINT（<機能>）
文　例	LX＝POINT(0)

解　説　最後にグラフィック操作の行われた座標（LP）を<機能>の指定により，スクリーン座標あるいは，ワールド座標で与えます.

<機能>は0から3の整数値をとり，それぞれの場合にPOINT関数の返す値の意味は次のとおりです.

0：最後にグラフィック操作の行われた点のX座標をワールド座標系上の値に変換して返す.

1：同じくY座標を返す.

2：最後にグラフィック操作の行なわれた点のX座標を，スクリーン座標系上の値に変換して返す.

3：同じくY座標を返す.

最後に操作の行われた座標がウィンドウの外であった場合には，POINT関数の返す座標に対応する点は，画面上には表示されていない点であることに注意してください．またこの時，スクリーン座標系に変換されて返された値である場合には，ビューポート外になることに注意してください.

参照：POINT(2章)，1章・ウィンドウとビューポート

サンプルプログラム

```
100 SCREEN 0,0:CLS 3:GOSUB *PRINTLP
110 PSET(100,100):GOSUB *PRINTLP
120 WINDOW(-100,-100)-(100,100)
130 PSET(50,50):GOSUB *PRINTLP:END
140 *PRINTLP:PRINT:PRINT "World  coordinate: ";
150  PRINT "X =";POINT(0),"Y =";POINT(1)
160  PRINT "Screen coordinate: ";
170  PRINT "X =";POINT(2),"Y =";POINT(3)
180 RETURN
```

```
World  coordinate: X = 0     Y = 0
Screen coordinate: X = 0     Y = 0

World  coordinate: X = 100   Y = 100
Screen coordinate: X = 100   Y = 100

World  coordinate: X = 50    Y = 50
Screen coordinate: X = 479   Y = 149
```

〔2〕POINT　N88

機　能	スクリーン座標上の指定された座標に表示されているドットの色を与える.
書　式	POINT (Sx, Sy)
文　例	C=POINT(473, 120)

解　説　(Sx, Sy)により指定されたスクリーン座標上に表示されているドットの色をパレット番号で返します．また，指定された座標 (Sx, Sy) がビューポートの外にある場合には，POINT 関数は－1を返します．

座標はスクリーン座標で指定することに注意してください．白黒モードの場合には，POINT 関数は，リセット（黒）またはセット（白）をドットの色として返します．

サンプルプログラム

```
100 '-- change color paintting --
110 DEF FNX=RND*639:DEF FNY=RND*199
120 DEF FNC=INT(RND*7+1)
130 SCREEN 0,0:CLS 3
140 '-- circle paint --
150 FOR I=1 TO 20
160  X=FNX:Y=FNY:R=RND*80+20:C=FNC
170  CIRCLE(X,Y),R,7:PAINT(X,Y),C,7
180 NEXT I
190 '-- chage color --
200 FOR I=1 TO 50
210  X=FNX:Y=FNY:C=FNC:CC=POINT(X,Y)
220  IF CC=0 OR CC=C THEN 240
230  PAINT(X,Y),C,7
240 NEXT I
```

POINT　Ⓝ

機能　指定した座標にドットがセットされているかいないかを与える.

書式　POINT（x, y）

文例　CL＝POINT(60, 60)

解説　指定したグラフィック座標に，ドットがセットされている場合には－1，リセットされていたり，キャラクタが表示されている場合には0を与えます.

サンプルプログラム

```
100 PRINT CHR$(12)
110 LINE(0,50)-(159,50),PSET
120 X=50:Y=0
130 IF POINT(X,Y)=-1 THEN END
140 PSET(X,Y)
150 Y=Y+1
160 GOTO 130
```

PORT　Ⓝ

機　能　**指定された RS-232C 通信ポートの入出力バッファに入力されている文字数を与えます.**

書　式　PORT(〈ポート番号〉)

文　例　PORT(1)

解　説　〈ポート番号〉で指定された RS-232C 通信ポートの入力バッファに現在何文字入力されているかを知る関数です. 本体内の RS-232C インタフエースを使用する場合〈ポート番号〉は 1 を指定します.

RS-232C 通信ポートの入出力バッファは127文字分あり, データがそのバッファの容量を越えた場合"Communication buffer overflow" エラーが起ります. したがって, PORT 関数でバッファがある程度いっぱいになったら INPUT% などでデータの読み出しを行ってください.

サンプルプログラム

```
100 '-- port sample --
110 'switch to RS-232C channel
120 OUT &H30,&H20 OR (&HF AND PEEK(&HEA66))
130 POKE &H8001,&H23:POKE &H8000,&H7F
140 OUT &HE6,4
150 OUT &HE4,&HFF
160 INIT %1,&HFA,&H37
170 IF PORT(1) THEN PRINT %1,INPUT$(PORT(1),%1);
180 GOTO 170
190 '*********************************
200 ' ﾌﾟﾛｸﾞﾗﾑ ｼｭｳﾘｮｳ ｺﾞ out &hE6,&h0
210 ' ｦ ｼﾞｯｺｳ ｼﾃ ｸﾀﾞｻｲ｡
220 '*********************************
```

POS　Ⓝ88 Ⓝ

機　能　テキスト画面上の現在のカーソルの水平位置を与える.

書　式　POS（〈式〉）

文　例　P＝POS（0）

解　説　式はダミーで通常 0 を使います.

与えられる値はキャラクタ座標の水平位置です.

参照：CSRLIN

サンプルプログラム

```
100 FOR I=32 TO 255
110  IF POS(0)=>32 THEN PRINT
120  PRINT CHR$(I);" ";
130 NEXT I

run
  ! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ ¥ ] ^ _
  a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~
▁ ▂ ▃ ▄ ▅ ▆ ▇ █ ▏ ▎ ▍ ▌ ▋ ▊ ▉ ┼
┴ ┬ ┤ ├ ▔ ─ │ ▕ ┌ ┐ └ ┘ ╭ ╮ ╰ ╯
  。 「 」 、 ・ ヲ ァ ィ ゥ ェ ォ ャ ュ ョ ッ
ー ア イ ウ エ オ カ キ ク ケ コ サ シ ス セ ソ
タ チ ツ テ ト ナ ニ ヌ ネ ノ ハ ヒ フ ヘ ホ マ
ミ ム メ モ ヤ ユ ヨ ラ リ ル レ ロ ワ ン ゛ ゜
═ ╞ ╪ ╡ ◢ ◣ ◥ ◤ ♠ ♥ ♦ ♣ ● ○ ╱ ╲
╳ 円 年 月 日 時 分 秒
Ok
```

R

RIGHT$ N88 N

機　能	文字列の右側から任意の長さの文字列を与える。
書　式	RIGHT$ (〈文字列〉, 〈数式〉)
文　例	PRINT RIGHT$ (A$, 4)

解　説　〈数式〉の値は0から255までの範囲になければなりません。

〈数式〉の値が0ならば，RIGHT$はヌルストリングを値とします。

〈数式〉が〈文字列〉の文字数より大きければ，〈文字列〉のすべてを値とします。

参照：LEFT$, MID$

サンプルプログラム

```
100 ' ﾓｼﾞﾚﾂ(10) ﾉ ｶｲﾃﾝ ( -> )
110  INPUT A$
120  FOR I=1 TO 11
130   PRINT A$
140   B$=RIGHT$(A$,1)
150   A$=B$+MID$(A$,1,LEN(A$)-1)
160  NEXT I
170 END

run
? 1234567890
1234567890
0123456789
9012345678
8901234567
7890123456
6789012345
5678901234
4567890123
3456789012
2345678901
1234567890
Ok
```

RND　N88 N

機　能	乱数を与える.
書　式	RND〔(<数式>)〕
文　例	R=RND(1)

解　説　0以上1未満の乱数を与えます. 発生する乱数は, RUN及びCLEARが実行されるごとに同系列を取ります. 但し, RANDOMIZE文によってその系列をかえることもできます.

発生される乱数は〈数式〉の値によって次のように異なります.

〈数式〉が負の場合――新しい乱数系列を設定します.

〈数式〉が0の場合――1つ前に発生した乱数の値をとります.

〈数式〉が正の場合――次の乱数を発生します.

〈数式〉が省略された場合は正の値として次の乱数を発生します.

N-BASICでは〈数式〉を省略することはできません.

参照：RANDOMIZE

サンプルプログラム

```
100 '-- ﾗﾝｽｳ ｹﾝﾃｲ --
110 DIM SUM(6)
120 FOR I=1 TO 100
130   DA=INT(RND*6+1)
140   SUM(DA)=SUM(DA)+1
150 NEXT I
160 FOR I=1 TO 6
170   PRINT I"-";SUM(I);"%"
180 NEXT I

run
 1 - 22 %
 2 - 15 %
 3 - 13 %
 4 - 12 %
 5 - 17 %
 6 - 21 %
Ok
```

SEARCH（ディスク） N88

機能　任意の配列変数の要素の中から指定された値を捜し出しその要素ナンバーを与える.

書式　SEARCH（〈配列変数名〉, 〈整数表記〉〔, 〈開始ナンバー〉〕〔, 〈ステップ値〉〕）

文例　NUM＝SEARCH(A%, 100, 0, 3)

解説　〈配列変数名〉で指定された配列変数の要素の中から〈整数表記〉で指定された値を捜し，最初に見つかった要素ナンバーを返します．もし指定された値がその配列変数内で見つからなかった場合は－1を値として返します．

〈配列変数名〉は整数型の一次元配列でなければならず，またこの配列変数はSEARCH関数の実行に先だってDIM文によって宣言されていなければなりません．

〈整数表記〉には配列要素の中から抽出したい値を指定します．もちろんこの値は整数でなければならず，もし実数を指定した場合は整数化してから実行します．

〈開始ナンバー〉と〈ステップ値〉はオプションであり省略可能です．〈開始ナンバー〉には，配列内のデータのどこから捜し始めるかを指定します．もし省略した場合は配列内データの最初から捜し始めます．但し，この値は現在OPTION BASEによって設定されている添字の底の大きさによって異なります．

〈ステップ値〉を指定した場合はそのステップをもってサーチします．省略した場合は1が採用されすべての要素についてサーチすることになります．

SEARCHはランダムアクセス・ファイルのインデックスのサーチやGET@，PUT@で用いる配列データのサーチなどに使うことができます．

サンプル
プログラム

```
100 DATA ｺｸｺﾞ,ｽｳｶﾞｸ,ｴｲｺﾞ
110 'data type : ｼｭｯｾｷﾊﾞﾝｺﾞｳ,ﾃﾝｽｳ ...
120 DATA 1,87,34,56
130 DATA 3,96,85,29
140 DATA 5,85,38,68
150 DATA 2,75,97,54
160 DATA 4,49,95,84
170 '
180 N=5:ITEM=4
190 DIM INDEX%(N*ITEM-1),ITEM$(3)
200 FOR I=1 TO 3:READ ITEM$(I):NEXT I
210 FOR I=0 TO N-1
220  FOR J=0 TO ITEM-1
230   READ INDEX%(I*ITEM+J)
240  NEXT J
250 NEXT I
260 '
270 *ENTRY
280 INPUT "ｼｭｯｾｷﾊﾞﾝｺﾞｳ(1-5)";SN
290 FAC=SEARCH(INDEX%,SN,,ITEM)
300 IF FAC=-1 THEN 280
310 FOR I=1 TO 3
320  PRINT ITEM$(I),INDEX%(FAC+I)
330 NEXT I
340 PRINT:GOTO *ENTRY

run
ｼｭｯｾｷﾊﾞﾝｺﾞｳ(1-5)? 2
ｺｸｺﾞ            75
ｽｳｶﾞｸ           97
ｴｲｺﾞ            54

ｼｭｯｾｷﾊﾞﾝｺﾞｳ(1-5)? 4
ｺｸｺﾞ            49
ｽｳｶﾞｸ           95
ｴｲｺﾞ            84

ｼｭｯｾｷﾊﾞﾝｺﾞｳ(1-5)? 1
ｺｸｺﾞ            87
ｽｳｶﾞｸ           34
ｴｲｺﾞ            56

ｼｭｯｾｷﾊﾞﾝｺﾞｳ(1-5)?
Break in 280
Ok
```

SGN　N88　N

機　能　符号を調べる.

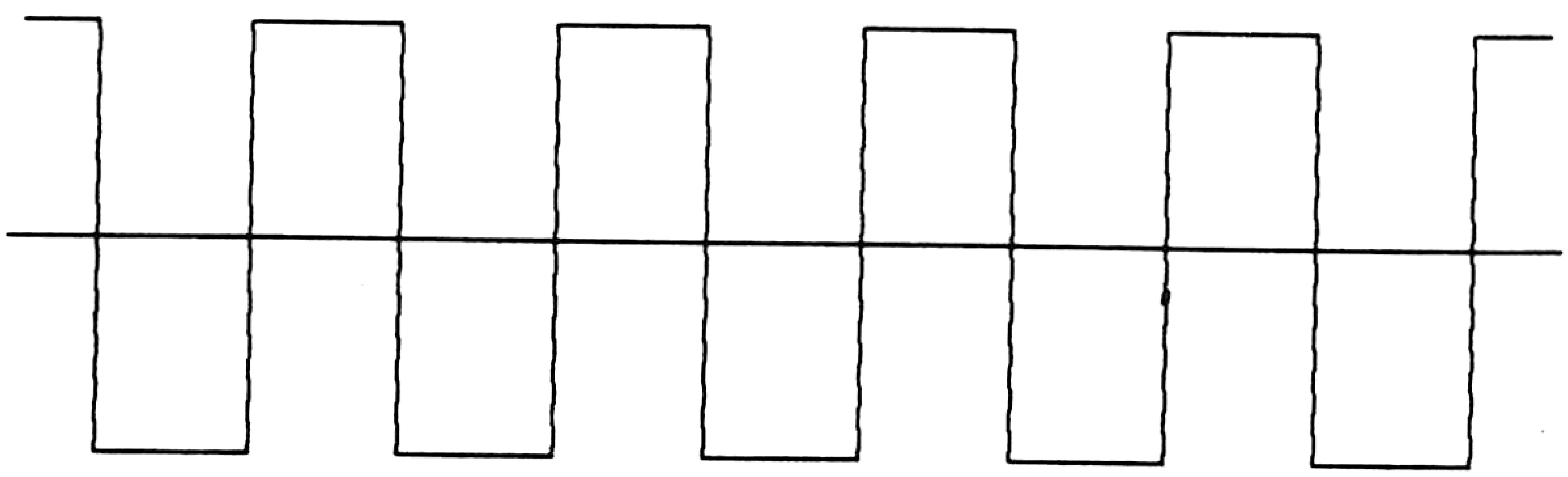
書　式　SGN (〈数式〉)

文　例　PRINT SGN (A)

解　説　〈数式〉が正の場合は 1 を,〈数式〉が 0 の場合は 0 を,〈数式〉が負の場合には, − 1 を与えます.

サンプルプログラム

```
100  SCREEN 0,0:CLS 3
110  LINE (0,100)-(639,100),1
120  X=5
130  FOR I=0 TO 31.42 STEP .05
140   S=SGN(INT(COS(I)*20))
150   ON S+2 GOSUB *PLUS,*ZERO,*MINUS
160   X=X+1
170  NEXT I
180 END
190 *PLUS
200  PSET (X,20),5:RETURN
210 *ZERO
220  LINE (X,20)-(X,179),5:RETURN
230 *MINUS
240  PSET (X,179),5:RETURN
```



SIN　N88　N

機　能　正弦値（サイン）を与える．

書　式　SIN（<数式>）

文　例　PRINT SIN（1）

解　説　<数式>の，単位をラジアンとした場合の三角関数サインの値を与えます．

<数式>に倍精度実数が含まれる場合は，倍精度の値を与えますが，他の場合には，単精度の値を与えます．

注意：倍精度機能は，N88DISK-BASICモードのみで有効です。

参照：COS，TAN

サンプルプログラム

```
100 ' csec ｦ ﾓﾄﾒﾙ
110  PI=3.1416
120  F$="csc(###°)=###.#####
130  INPUT "ｶｸﾄﾞ :";D
140  R=D MOD 180
150   IF R=0 THEN PRINT "ﾃｲｷﾞｻﾚﾏｾﾝ":END
160  C=1/SIN(R/180*PI)
170  PRINT USING F$;D,C
180 END

run
ｶｸﾄﾞ :? 15
csc( 15°)=  3.86369
Ok
run
ｶｸﾄﾞ :? 60
csc( 60°)=  1.15470
Ok
```

SPACE$　N88 N

機　能	任意の長さの空白文字列を与える.
書　式	SPACE$ (〈数式〉)
文　例	PRINT SPACE$ (10) ; "ABC"

解　説　〈数式〉の数だけの空白をもった文字列を与えます. その値は 0 から 255までの範囲になければなりません.

参照：SPC

サンプルプログラム

```
100 '-- squre curve --
110  FOR I=-6 TO 6
120   A$=CHR$(&H88)+SPACE$(ABS(I*I))+"*"
130   PRINT A$
140  NEXT I
150 END

run
|                                    *
|                         *
|                *
|         *
|    *
| *
|*
| *
|    *
|         *
|                *
|                         *
|                                    *
Ok
```

SPC　N88　N

機　能　任意の数だけ空白を出力する．

書　式　SPC（〈数式〉）

文　例　PRINT SPC（10） A$

解　説　SPC 関数は PRINT 文や LPRINT 文など出力文中でのみ使用することができます．

〈数式〉の値は，−32768から32767まで許されますが，負の数はすべて 0 と見なされます．また正の数の場合でも，現在の水平表示文字数以上の場合は，〈数値〉をその水平表示文字数で割った余りを値とします．

参照：SPACE$，TAB

サンプルプログラム

```
100  FOR I=0 TO 10
110   PRINT "N88-BASIC";SPC(I);"N-BASIC"
120  NEXT I
130 END

run
N88-BASICN-BASIC
N88-BASIC N-BASIC
N88-BASIC  N-BASIC
N88-BASIC   N-BASIC
N88-BASIC    N-BASIC
N88-BASIC     N-BASIC
N88-BASIC      N-BASIC
N88-BASIC       N-BASIC
N88-BASIC        N-BASIC
N88-BASIC         N-BASIC
N88-BASIC          N-BASIC
Ok
```

SQR　N88　N

機　能	平方根（スクエアールート）を与える.
書　式	SQR（〈数式〉）
文　例	A＝SQR（2）

解　説　〈数式〉で指定された値の平方根を与えます.

〈数式〉に倍精度実数が含まれる場合は，倍精度の値を与えますが，他の場合には単精度の値を与えます.

注意：倍精度機能は，N88DISK-BASICモードのみ有効です.

サンプルプログラム

```
100 '-- square root table --
110 PRINT " << SQUARE ROOT TABLE >>"
120 FOR N=0 TO 10
130   PRINT "N =";N,"ROOT(N) =";
140   PRINT USING "##.#####";SQR(N)
150 NEXT N

run
 << SQUARE ROOT TABLE >>
N = 0          ROOT(N) = 0.00000
N = 1          ROOT(N) = 1.00000
N = 2          ROOT(N) = 1.41421
N = 3          ROOT(N) = 1.73205
N = 4          ROOT(N) = 2.00000
N = 5          ROOT(N) = 2.23607
N = 6          ROOT(N) = 2.44949
N = 7          ROOT(N) = 2.64575
N = 8          ROOT(N) = 2.82843
N = 9          ROOT(N) = 3.00000
N = 10         ROOT(N) = 3.16228
Ok
```

STR$ N88 N

機能	数値を表わす文字列を与える.
書式	STR$ (〈数式〉)
文例	A$=STR$ (123)

解説　〈数式〉によって指定された値を文字に変換します．但し，数値及び変数以外を指定してはいけません．〈数式〉の値は整数型，実数型，（倍精度，単精度）及び浮動小数点など，すべての数値型に有効です．変換された文字は，先頭に1文字分のスペースを含みます．

注意：STRING$関数と区別すること．

参照：VAL，STRING$

サンプルプログラム

```
100 ' 00:00:00 カラ シテイシタ ビョウスウ ダケ
110 ' ケイカシタ ジコク ヲ モジレツ デ モトメル
120  DEF FNST$(A$)=RIGHT$("0"+RIGHT$(A$,LEN(A$)-1),2)
130  INPUT "秒スウ :",SS
140  H=SS¥3600:M=(SS-H*3600)¥60
150  S=SS MOD 60
160  H$=STR$(H):M$=STR$(M):S$=STR$(S)
170  H$=FNST$(H$):M$=FNST$(M$):S$=FNST$(S$)
180  PRINT "00:00:00 カラ ";SS;"秒 ケイカシタ";
190  PRINT " ジコクハ ";H$;":";M$;":";S$;" デス"
200 END

run
秒スウ :10000
00:00:00 カラ  10000 秒 ケイカシタ ジコクハ 02:46:40 デス
Ok
```

STRING$ N88 N

機　能　任意の文字を任意の数だけ与える．

書　式　STRING$（<式>，| <文字式> | ）
　　　　　　　　　　　　　　　| <数式> |

文　例　A$＝STRING$（10，〝+〞）

解　説　与える文字は，〈文字式〉または〈数式〉によって指定します．〈文字式〉の場合は，与える〈文字列〉の最初の一文字が有効です．〈数式〉の場合は，値が，0から255の範囲に限られ，この値をキャラクタコードとみなし，対応する文字を採用します．

参照：STR$

サンプルプログラム

```
100   INPUT A$
110   L=LEN(A$)
120   PRINT STRING$(L+4,"+")
130   PRINT "+ ";SPC(LEN(A$)+1);"+"
140   PRINT "+ ";A$;" +"
150   PRINT "+ ";SPC(LEN(A$)+1);"+"
160   PRINT STRING$(L+4,"+")
170  END
```

```
run
? STRING statement sample program
++++++++++++++++++++++++++++++++++++
+                                  +
+ STRING statement sample program +
+                                  +
++++++++++++++++++++++++++++++++++++
OK
```

TAB　N88　N

機　能	現在のカーソルのある行の任意の位置まで空白を出力する．
書　式	TAB（<数式>）
文　例	PRINT TAB（10）；〝PC-8801MK2〃

解　説　TABはPRINT文やLPRINT文など出力文中のみで使用されます．

〈数式〉の値は，－32768から32767（0が左端）まで許されますが，負の数はすべて0と見なされます．また正の数の場合でも，現在の水平表示文字数以上の場合は，〈数値〉をその水平表示文字数で割った余りを値とします．

SPCとの区別に注意．

参照SPC

T

サンプルプログラム

```
100  FOR DEG=0 TO 180 STEP 20
110   RAD=DEG/180*3.14159
120   PRINT "DEG =";DEG;
130   PRINT TAB(15)"SIN(DEG) =";SIN(RAD);
140   PRINT TAB(40)"COS(DEG)=";COS(RAD)
150  NEXT DEG
160 END

run
DEG = 0         SIN(DEG) = 0            COS(DEG)= 1
DEG = 20        SIN(DEG) = .34202       COS(DEG)= .939693
DEG = 40        SIN(DEG) = .642787      COS(DEG)= .766045
DEG = 60        SIN(DEG) = .866025      COS(DEG)= .500001
DEG = 80        SIN(DEG) = .984808      COS(DEG)= .173649
DEG = 100       SIN(DEG) = .984808      COS(DEG)=-.173647
DEG = 120       SIN(DEG) = .866027      COS(DEG)=-.499998
DEG = 140       SIN(DEG) = .642789      COS(DEG)=-.766043
DEG = 160       SIN(DEG) = .342023      COS(DEG)=-.939692
DEG = 180       SIN(DEG) = 2.8088E-06   COS(DEG)=-1
Ok
```

TAN　N88 N

機　能	**正接値(タンジェント)を与える.**
書　式	TAN (<数式>)
文　例	A=TAN(123)

解　説　<数式>の単位をラジアンとした場合の三角関数タンジェントの値を与えます.

<数式>に倍精度実数が含まれる場合は，倍精度の値を与えますが，他の場合には，単精度の値を与えます.

注意：倍精度機能は，N88DISK-BASICモードのみ有効です.

参照：SIN，COS

サンプルプログラム

```
100 ' cot ヲ モトメル
110  PI=3.1416
120  F$="cot(###°)=###.######
130  INPUT "カクト゛ :";D
140  R=D MOD 180
150   IF R=0 THEN PRINT "テイキ゛サレマセン":END
160  C=1/TAN(R/180*PI)
170  PRINT USING F$;D,C
180 END

run
カクト゛ :? 45
cot( 45°)=  1.000000
Ok
run
カクト゛ :? 30
cot( 30°)=  1.73205
Ok
run
カクト゛ :? 23
cot( 23°)=  2.35585
Ok
```

TIME$ N88 N

機　　能	内蔵クロックの時刻を与える.
書　　式	1) TIME$
	2) TIME$＝"hh : mm : ss"
文　　例	PRINT TIME$

解　　説　TIME$はシステム変数であり現在の時刻を持っています.

TIME$の形式は,

　　HH：MM：SS

で示されます.

また, TIME$は次のように新たに時刻を設定することもできます. HH は00から23まで, MM は00から59まで, SS は00から59までの数です.

　　TIME$＝ "12：30：00"

参照：DATE$

サンプルプログラム

```
100 ' シテイ ジカン ゴ ノ ジコク ヲ モトメル
110 T$=TIME$
120 PRINT "ゲンザイ ノ ジコク ハ ";T$;"デス"
130 INPUT " ジカン (0-23):",H
140 INPUT " フン   (0-59):",M
150 INPUT " ビョウ (0-59):",S
160 HH=VAL(LEFT$(T$,2))
170 MM=VAL(MID$(T$,4,2))
180 SS=VAL(RIGHT$(T$,2))
190 SS=SS+S:MM=MM+M:HH=HH+H
200 FOR I=0 TO 1
210  MM=MM+SS¥60:SS=SS MOD 60
220  HH=HH+MM¥60:MM=MM MOD 60
230  DD=DD+HH¥24:HH=HH MOD 24
240 NEXT
250 PRINT "モトメル ジコク ハ"
260  IF DD=1 THEN.PRINT "ヨクジツ ノ";
270 PRINT USING " ###時 ###分 ###秒 デス";HH,MM,SS
280 END

run
ゲンザイ ノ ジコク ハ 04:38:04デス
 ジカン (0-23):4
 フン   (0-59):30
 ビョウ (0-59):15
モトメル ジコク ハ
  9時  8分 19秒 デス
Ok
```

USR N88 N

機　能	機械語で作られた関数ルーチンを呼び出す.
書　式	USR〔<番号>〕(<引数>)
文　例	I=USR3 (J)

解　説　ユーザーの作成した機械語関数ルーチン（ユーザー関数）を呼び出します．ユーザー関数は，予め，メモリ中に準備されており，また DEF USR 文により，その実行開始番地が設定されていなければなりません．

〈番号〉は 0 から 9 までの値で，DEF USR 文により定義された番号に対応します．〈番号〉が省略された場合には，0 と解釈されます．

〈引数〉により BASIC から機械語ルーチンへの値の受け渡しをすることができます．〈引数〉の受け渡し方法については，APPENDIX・機械語プログラムの作り方を参照してください．

N88DISK-BASICでは，機械語ルーチンを呼び出す方法としてUSR関数の他にCALL文が用意されています．CALL文では複数の引数を機械語プログラムに受け渡すことができます．

U

参照：DEF USR，CALL，APPENDIX・機械語プログラムの作り方

サンプルプログラム

```
100 CLEAR ,&HDFFF
110 DEF USR =&HE000
120 DEF USR1=&HE200
130 POKE &HE000,&HC9
140 POKE &HE200,&HC9
150 PRINT USR(100)
160 PRINT USR1(1.23)

run
 100
 1.23
Ok
```

VAL　N88　N

機　能	**文字列の表す数値を与える.**
書　式	VAL (<文字列>)
文　例	A=VAL ("2")

解　説　<文字列>を数値に変換して与えます. もし, <文字列>の最初の文字が+, −, ., &, または数字でなければ, VAL の値は 0 になります.

但し, 文字列中のスペースは無視します.

参照 : STR$

サンプルプログラム

```
100 ' 00:00:00 ｶﾗﾉ ｹﾞﾝｻﾞｲﾉ
110 ' ｼﾞｺｸ ﾏﾃﾞﾉ ﾋﾞｮｳｽｳ ｦ ﾓﾄﾒﾙ
120  T$=TIME$
130  H=VAL(LEFT$(T$,2))
140  M=VAL(MID$(T$,4,2))
150  S=VAL(RIGHT$(T$,2))
160  SS=H*3600+M*60+S
170  PRINT "00:00:00 ｶﾗ ";T$;" ﾏﾃﾞﾊ";
180  PRINT SS;"秒ｶﾝ ﾃﾞｽ"
190 END

run
00:00:00 ｶﾗ 04:43:22 ﾏﾃﾞﾊ 17002 秒ｶﾝ ﾃﾞｽ
Ok
```

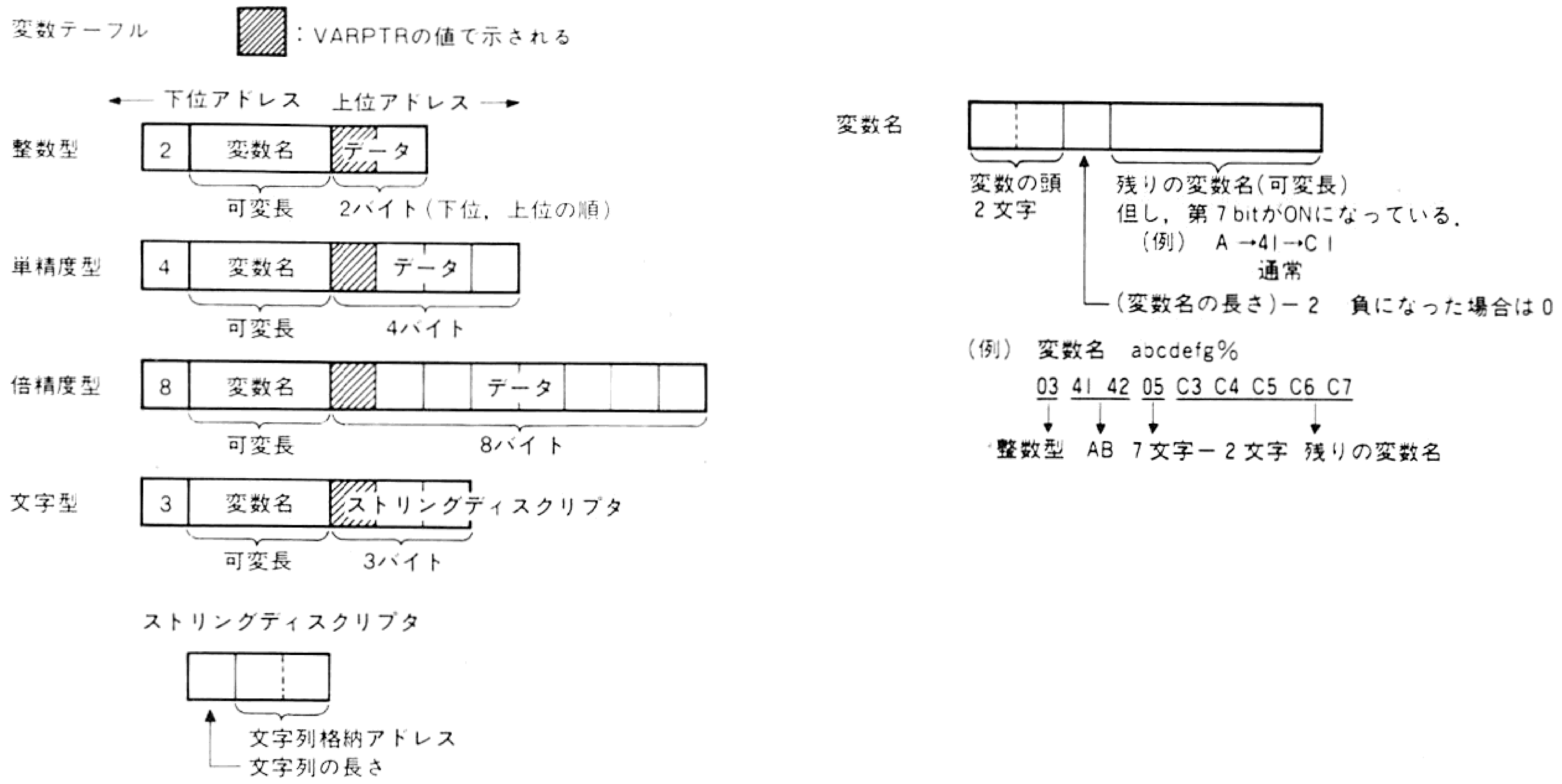
V

VARPTR　N88　N

機　能	変数の格納されているメモリ番地，ファイルに割り当てられているバッファの開始番地を与える．
書　式	1) VARPTR (〈変数名〉) 2) VARPTR (# 〈ファイル番号〉)
文　例	1) PRINT HEX$ (VARPTR (A)) 2) BVAD=VARPTR (#1)

解　説

1) 〈変数名〉で指定した変数のデータが格納されている，変数領域のメモリ番地を与える．このとき指定される変数には値がついていなければなりません．



2)指定した〈ファイル番号〉に割り当てられている入出力バッファの開始番地を与える．

サンプルプログラム

```
100 '-- (local value) ¥ 2 ヲモトメル --
110  DEFINT A
120  INPUT "Value";A
130  ADR=VARPTR(A)
140  AL=(A¥2) MOD 256
150  AH=(A¥2)¥256
160  POKE ADR,AL
170  POKE ADR+1,AH
180  PRINT A
190  GOTO 120
```

```
run
Value? 1245
 622
Value? 653
 326
Value? 9035
 4517
Value?
Break in 120
Ok
```

VIEW　N88

機　能	現在のビューポートの設定位置を与える.
書　式	VIEW (<機能>)
文　例	SX1＝VIEW (0)

解　説　VIEW 文により設定されている現在のビューポートの位置 (Sx_1, Sy_1)－(Sx_2, Sy_2) を返す関数です. <機能> は, 0 から 3 までの値をとり, それぞれの値のときにとる関数値の意味は次のとおりです.

0 : ビューポートの左上の頂点のX座標 (Sx_1)
1 : ビューポートの左上の頂点のY座標 (Sy_1)
2 : ビューポートの右下の頂点のX座標 (Sx_2)
3 : ビューポートの右下の頂点のY座標 (Sy_2)

VIEW 関数は, ディスプレイ画面上のビューポートの位置を返す関数ですから, 返される値は, オリジナルスクリーン座標であることに注意してください.

参照 : VIEW (2章), MAP, WINDOW (2章, 3章), 1章・ウィンドウとビューポート

サンプルプログラム

```
100 SCREEN 0,0
110 GOSUB *VIEWINFO
120 WINDOW(-100,-100)-(100,100)
130 GOSUB *VIEWINFO
140 VIEW(100,50)-(300,150)
150 GOSUB *VIEWINFO:END
160 *VIEWINFO
170  PRINT "SX1 :";VIEW(0),
180  PRINT "SY1 :";VIEW(1)
190  PRINT "SX2 :";VIEW(2),
200  PRINT "SY2 :";VIEW(3):PRINT
210 RETURN

run
SX1 : 0        SY1 : 0
SX2 : 639      SY2 : 199

SX1 : 0        SY1 : 0
SX2 : 639      SY2 : 199

SX1 : 100      SY1 : 50
SX2 : 300      SY2 : 150

Ok
```

WINDOW　N88

機　　能	現在のウィンドウの設定位置を与える.
書　　式	WINDOW (〈機能〉)
文　　例	WY1＝WINDOW(1)

解　　説　WINDOW 文により設定されている現在のウインドウの位置 $(Wx_1, Wy_1)-(Wx_2, Wy_2)$ を返す関数です. 〈機能〉は, 0から3までの値をとり, それぞれの値のときにとる関数値の意味は次のとおりです.

0：ウィンドウの左上の頂点のX座標 (Wx_1)
1：ウィンドウの左上の頂点のY座標 (Wy_1)
2：ウィンドウの右下の頂点のX座標 (Wx_2)
3：ウィンドウの右下の頂点のY座標 (Wy_2)

WINDOW 関数の返す値は, ワールド座標であることに注意してください.

参照：VIEW (2章, 3章), MAP, WINDOW (2章), 1章・ウインドウとビューポート

サンプルプログラム

```
100 SCREEN 0,0
110 GOSUB *WINDOWINFO
120 VIEW(100,50)-(300,150)
130 GOSUB *WINDOWINFO
140 WINDOW(-100,-100)-(100,100)
150 GOSUB *WINDOWINFO:END
160 *WINDOWINFO
170  PRINT "WX1 :";WINDOW(0),
180  PRINT "WY1 :";WINDOW(1)
190  PRINT "WX2 :";WINDOW(2),
200  PRINT "WY2 :";WINDOW(3):PRINT
210 RETURN

run
WX1 : 0         WY1 : 0
WX2 : 639       WY2 : 199

WX1 : 100       WY1 : 50
WX2 : 300       WY2 : 150

WX1 :-100       WY1 :-100
WX2 : 100       WY2 : 100

Ok
```

W

第4章

拡張命令

注意：拡張命令は，拡張命令パッケージを追加したときに使えます。

CMD CLS Ⓝ88

機　能　**現在アクティブな画面または，指定したグラフィック画面をクリアする．（高速版）**

書　式　CMD CLS〔<機能>〕

文　例　CMD CLS 3

解　説　〈機能〉は，1，2，3および9～15の値をとり，それぞれ次のように働きます。省略された場合は1が選択されます．

1）　テキスト画面のみをクリアします．

2）　現在アクティブなグラフィック画面をクリアします．グラフィック表示がSCREEN文によりカラーモードに設定されている場合には，COLOR文により指定されたバックグラウンドカラーでクリアします．CLS文と異なるのは，ビューポートが無視されることと，LPが移動しないということです．

3）　テキスト画面，グラフィック画面をクリアします．
(CMD　CLS　1：CMD　CLS　2）と同じです．

9～15）　指定したグラフィック画面のみをクリアします．その画面がアクティブかどうかは関係しません．

	PAGE 0(B)	PAGE 1(R)	PAGE 2(G)
9	○	×	×
10	×	○	×
11	○	○	×
12	×	×	○
13	○	×	○
14	×	○	○
15	○	○	○

○ ……クリアする．　　× ……そのまま．

テキスト画面がクリアされる時は，CONSOLE文によって指定されたスクロールウィンドウ内のみをクリアします．

参照：SCREEN, COLOR, CONSOLE, CLS

サンプルプログラム

```
100 '--- CMD CLS sample ---
110 SCREEN 0,0
120 COLOR ,0 : CMD CLS 3
130 FOR I=0 TO 4
140   GOSUB *DRAW.LINE
150   CMD CLS 2
160 NEXT I
170 END
180 *DRAW.LINE
190 FOR J=1 TO 30
200   LINE-(RND*639,RND*199),INT(RND*7)+1
210 NEXT J
220 RETURN
```

CMD CUT N88

機　能　**拡張命令パッケージを切り離す.**

書　式　CMD CUT

文　例　CMD CUT

解　説　拡張命令パッケージを切り離し，専有していたエリアをフリーエリアとして開放します.

このステートメントを実行すると，CLEAR，&HE5FF も同時に行われます. 以後，CMD ステートメントは使えなくなります.

サンプルプログラム

```
cmd cut
Ok
```

CMD SING N88

機能	**音楽を奏でる.**
書式	CMD SING 〈サブコマンドストリング〉
文例	CMD SING "T150 04 RP2〔CDE〕EDC"
解説	PC-8801MKIIの内蔵スピーカーを利用して，サブコマンドストリングで指定されたメロディーを演奏します．サブコマンドストリングは，ミュージックサブコマンドを並べた文字列で，文字式も使えます．

ミュージックサブコマンドで指定できるのは，テンポ，オクターブ，長さ，音符，休符で，さらに，テキスト画面のON/OFF機能，リピート機能があります．

ミュージックサブコマンドは，文字とパラメータで指定します．パラメータは，数字(整数)または，変数名を与えます．
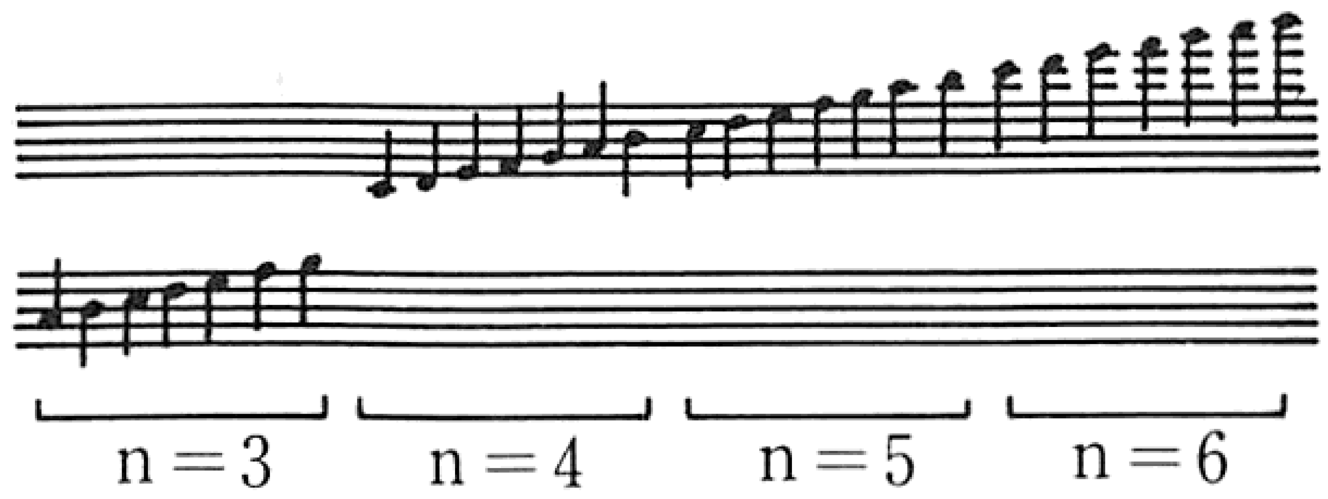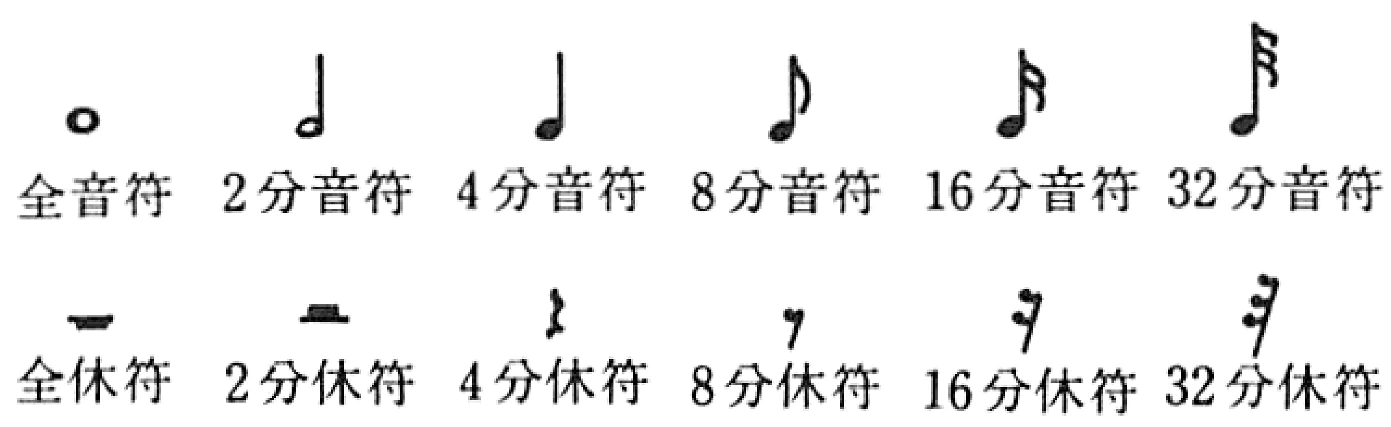
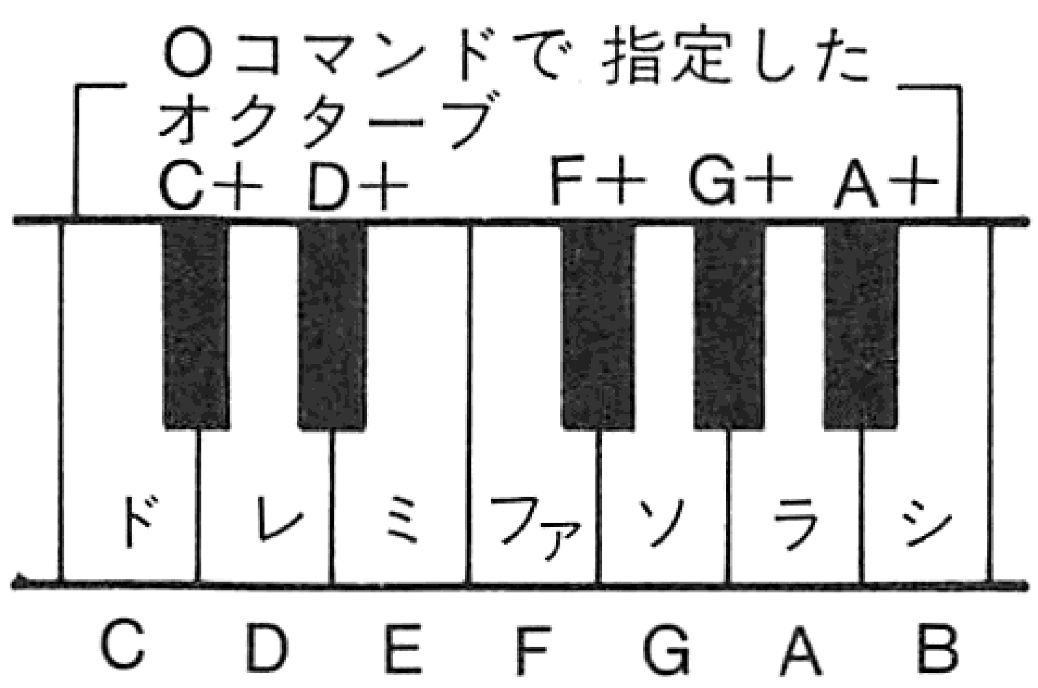
例1) "T120"

例2) "T(temp)" ：tempは変数名で，temp＝120のときは，"T120"と同じ．

注意：テキスト画面を消さないでCMD SINGを実行すると聞きづらい音になりますので，サブコマンドストリングの先頭にX0を付けるか，CMD TEXT OFFを実行して音を出すことをおすすめします．

また，音の高さ，長さなどは，テキスト画面を消したときを基準に設定してありますが，多少の誤差があります．

ミュージックサブコマンド

コマンド	条　件	機　　　能
Tn	48≦n≦255	テンポを設定します。数が大きいほど速くなります．　nの値で，1分間に演奏する4分音符の数を指定します。
On	3≦n≦6	オクターブを設定します．nの値とC～Bで演奏するオクターブの対応は，次のとおりです． 
Ln	1≦n≦32	音符，休符の長さのデフォルト値を設定します．nの値で，何分音符（休符）かを指定します． n＝1　n＝2　n＝4　n＝8　n＝16　n＝32 
R[n][.]	1≦n≦32	休符．nの値で，休符の長さを指定します．指定の方法は，Lコマンドと同じです．省略した場合は，Lコマンドで指定した値とします．符点休符には，“.”（ピリオド）を付けます．
C～B [±][n][.]	1≦n≦32	音符．C～Bで音符を指定します．“＋”で半音上がり，“－”で半音下がります．  nの値で，音符の長さを指定します．符点音符には，“.”（ピリオド）を付けます．
Xn	n＝0または n＝1	テキスト画面の表示を制御します．X0で，テキスト画面を消し，X1でもとに戻します．澄んだ音を出すときは，最初にX0を行います．
RPj[…]	1≦j≦255	[…]をj回くりかえします． ネスティングは，8レベルまでです．

＊　コマンドは，小文字でもかまいません．

サンプルプログラム

```
100 '--- CMD SING sample ---
110 CMD SING "X0T180O5L4"
120 A$="GB-2O6CD4.E8DC2O5AF4.G8A"
130 B$="O6F2F8.R16F4.E8DC2O5AF4.G8A"
140 CMD SING A$+"B-2GG4.F+8GA2F+D2"
150 CMD SING A$+"B-4.A8GF+4.E8F+G2G8.R16G2."
160 CMD SING B$+"B-2GG4.F+8GA2F+D2."
170 CMD SING B$+"B-4.A8GF+4.E8F+G2G8.R16G2."
180 END
```

CMD TEXT ON/OFF N88

機能	**テキスト画面の表示を制御する.**
書式	1) CMD TEXT ON
	2) CMD TEXT OFF
文例	CMD TEXT OFF
解説	テキスト画面の表示を止めたり，もとに戻すステートメントです.

1) テキスト画面を表示します.

2) テキスト画面の表示を止めます。これにより，プログラムの実行速度をアップすることができます.

コマンド待ちに戻る前（プログラムの終了時，プログラム中でのエラー発生時）には，自動的にテキスト画面がもとに戻ります.

なお，この機能は，CMD SINGのサブコマンドでもサポートされています.

注意：CMD TEXT ONなど（CMD SING “X1”, CMD CUT）によって，テキスト画面の再表示を行った場合，SCREEN文のパラメータ4がリセットされます.（SCREEN文参照）

サンプルプログラム

```
100 '--- CMD TEXT ON/OFF sample ---
110 TIME$="00:00:00"
120 GOSUB *CALC
130 PRINT "TEXT ON ... ";TIME$
140 TIME$="00:00:00"
150 CMD TEXT OFF
160 GOSUB *CALC
170 CMD TEXT ON
180 PRINT "TEXT OFF... ";TIME$
190 END
200 *CALC
210 FOR I=0 TO 999
220   A=SIN(I)
230 NEXT I
240 RETURN
```

CMD TURTLE　N88

機能　**タートルグラフィックで図形を描く.**

書式　CMD TURTLE <サブコマンドストリング>

文例　CMD TURTLE "PC3 RP6〔FD50RT60〕" ＋T$

解説　タートルの動きをグラフィックサブコマンドで指定しながら図形を描くステートメントです．サブコマンドストリングは，グラフィックサブコマンドを並べた文字列で文字式も使えます．

グラフィックサブコマンドは，文字とパラメータで指定します．パラメータは，数字(整数)または，変数名を与えます．

例1)　"FD 100"

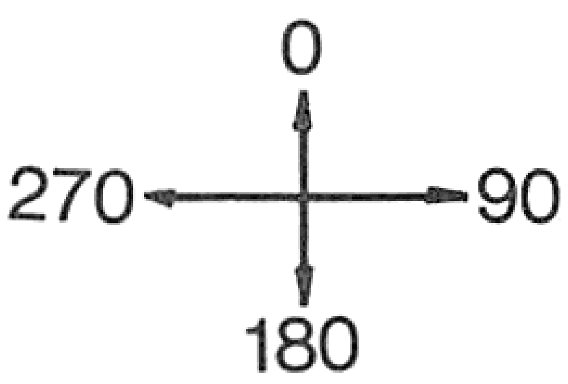
例2)　"FD(dist)"　：distは，変数名で，dist＝100のときは，"FD100"と同じ．

座標系は，PUT @，GET @文と同様にスクリーン座標系を用います．

タートルの位置は，LP(Last Reference Point)の位置です。タートルが移動すると，LPも変化します．

注意：STコマンドを使ってタートルを表示させた状態で，他のグラフィック命令(CLS 2)などを使うと，次にCMD TURTLEを実行した際，ゴミが出ることがあります．CMD TURTLEと他のグラフィック命令を混ぜて使う場合は，HTコマンドでタートルを消して使うようにしてください．

グラフィックサブコマンド

コマンド	条　件	機　　　　　能
FDn	−32768≦n ≦32767	タートルをn進めます．負の数を指定すると後退します．タートルの向きは変化しません．
BKn	−32768≦n ≦32767	タートルをn後退させます　負の数を指定すると前進します．タートルの向きは変化しません．
MVx,y	−32768≦x, y ≦32767	タートルをスクリーン座標（x, y）へ移動させます．タートルの向きは変化しません．
SXx	−32768≦x ≦32767	タートルをスクリーン座標(x,(現在のy座標))へ移動させます．(水平移動) タートルの向きは変化しません．
SYy	−32768≦y ≦32767	タールをスクリーン座標((現在のx座標), y)へ移動させます．(垂直移動) タートルの向きは変化しません．
HDn	−32768≦n ≦32767	タートルの向きを決めます． 正の数で時計まわり． 負の数で反時計まわり． 
LTn	−32768≦n ≦32767	タートルの向きを現在向いている方向からn度左に向けます．
RTn	−32768≦n ≦32767	タートルの向きを，現在向いている方向からn度右に向けます．
CPn	n＝0または n＝1	640×200モードで，x方向の補正をします．FD, BKコマンドで有効です．通常は，CP1(補正する)の状態で使います．
PD	なし	タートルのペンを下げます．このあとPUコマンドが実行されるまで，タートルは軌跡を残しながら移動します．
PU	なし	タートルのペンを上げます．タートルは軌跡を残さずに移動します．
PCn	0≦n≦7	タートルのペンの色を指定します． nは，パレット番号です．
HT	なし	タートルを消します．
ST	なし	タートルを表示します．
RPj[…]	1≦j≦255	[…]をj回くり返します． ネスティングは8レベルまでです．

※　コマンドは，小文字でもかまいません．

サンプル
プログラム

```
100 '--- CMD TURTLE sample ---
110 SCREEN 0,0 : CMD CLS 2 : CMD TEXT OFF
120 CMD TURTLE "ST PU MV320,100 PD"
130 FOR I=1 TO 200
140   CMD TURTLE "FD(i) RT92"
150 NEXT I
160 CMD TURTLE "HT"
170 END
```

APPENDIX

A　N88-BASICのプログラミング

ここでは N88-BASIC の主な機能の使い方をサンプル・プログラムを中心にプログラミングの立場で説明します．

A.1　白黒モードでのグラフィック

N88-BASIC の白黒モードでは，640×200ドットの画面を 3 ページ取ることができ，任意のページに書き込んだり表示したりすることができます．これをうまく使うと表ページを見せながら裏ページに書き込んだり，既に各ページに書き込んである図形を重ね合わせたりして，より効果的な画面を作ることができます．

ここでは各ページに SIN，COS，SIN＋COS のグラフをそれぞれ描き，描き終った時点で各ページを順に表示するプログラムを例にします．

160行までは初期設定であり，ここでスクリーンをクリアしています．次に軸を書き込みます．

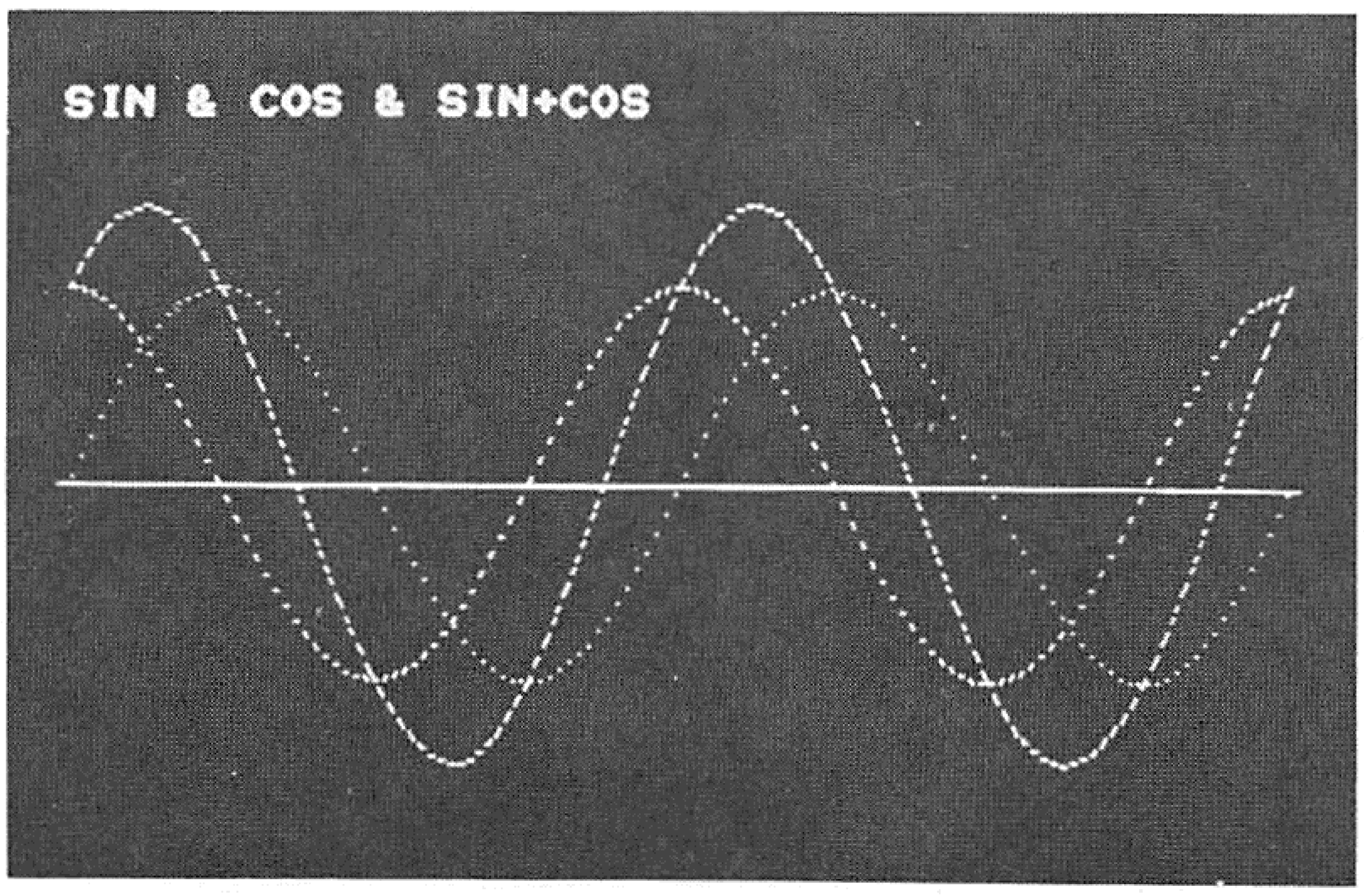
230行から340行のループで各ページに各三角関数のグラフを書き込んでいます．この状態ではすべてのページを見ることができます．

360行から420行のループは，SCREEN の〈ディスプレイページ〉をインクリメントしていくことによって各ページを色々な組み合わせで表示しています．

この 3 ページの他にテキスト画面をうまく使うとさらに効果的な表示とすることができます．

```
100 '-- B/W mode sample program --
110 '
120  DEFINT A-Z:DEFSNG I
130  DIM T$(7)
140  WIDTH 40,25:COLOR 7,,,7
150  CONSOLE 0,25,0,1
160  SCREEN 0,0:CLS 3
170 '
180  LINE(0,100)-(639,100),7
190  SCREEN 1,,0,7
200  FOR I=1 TO 7:READ T$(I):NEXT I
210  XO=5:Y1=100:Y2=50:Y3=50
220 '
230  FOR I=.1 TO 6.28*2+.1 STEP .1
240    SCREEN 1,,0,7
250    YS=100-SIN(I)*50
260    LINE(XO,Y1)-(XO+5,YS),7,,&H1010
270    SCREEN 1,,1,7
280    YC=100-COS(I)*50
290    LINE(XO,Y2)-(XO+5,YC),7,,&H3030
300    SCREEN 1,,2,7
```

```
310     YP=100-(SIN(I)+COS(I))*50
320     LINE(XO,Y3)-(XO+5,YP),7,,&HCCCC
330     XO=XO+5:Y1=YS:Y2=YC:Y3=YP
340  NEXT I
350 '-- paging --
360  FOR K=0 TO 20
370     FOR I=1 TO 7
380       SCREEN 1,,,I:LOCATE 0,0
390       PRINT LEFT$(T$(I)+SPACE$(16),19)
400       FOR J=0 TO 500:NEXT J
410     NEXT I
420  NEXT K
430  END
440 '-- index data --
450 DATA SIN,COS,SIN & COS,SIN+COS,SIN & SIN+COS
460 DATA COS & SIN+COS,SIN & COS & SIN+COS
```



A.2 ペイント（タイリング）の使い方

タイリングは〝タイル〟の大きさとパターンを適当に設定することによって，服地の模様のシミュレーションや中間色の表現など広範囲な応用が考えられます．

ここでは中間色を表示するプログラムを例に，タイリングの基本的なプログラミングを説明します．

120行から170行までは初期設定です．170行で作っている文字列はタイリングの際のバックグラウンド・ストリングに使うもので，現在のバックグラウンドカラー（黒）を表す文字列になっています．通常バックグラウンドカラーの上へのタイリングでは，バックグラウンド・ストリングの指定は省略できますが，説明のために設定しました．

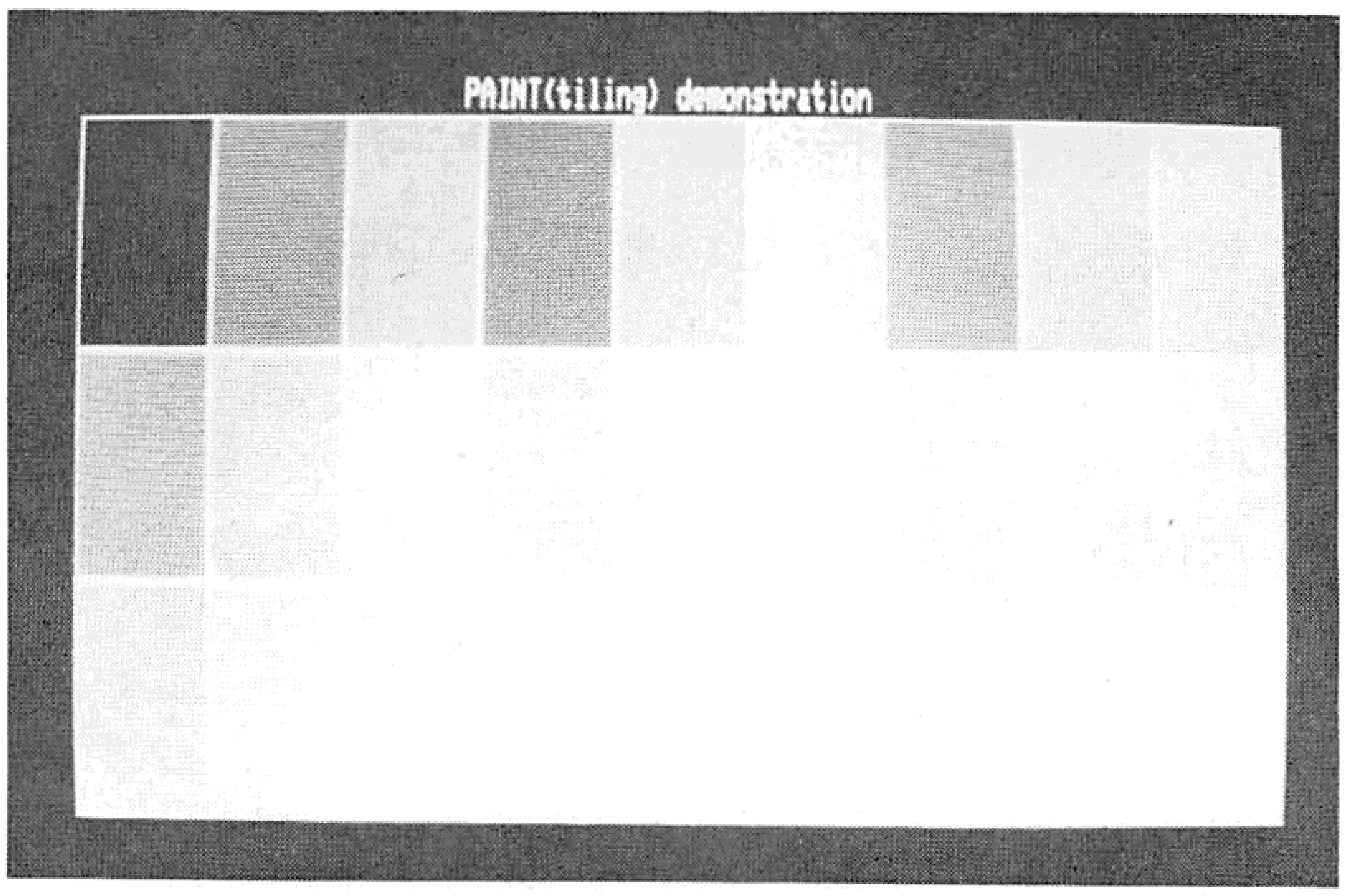
200行から330行までのループは各プレーンごとの色の比重を変えながら画面に

表示しています.

タイルストリングの生成は360行以降のサブルーチンで行われており,各プレーンの重みによってビットをセットしています(380行から410行).

ここで用いたのは3文字(1×8ドット)のタイルストリングですが,大きなストリングを用いれば様々な模様を表現することができます.

```
100 '--  PAINT(tiling) sample program  --
110 '
120  SCREEN 0,0:COLOR 7,0,0,7
130  CONSOLE 0,25,0,1:WIDTH 80,25
140  CLS 3:WINDOW(0,0)-(639,199)
150  DEFINT A-Z:DIM C(2),TILE(2)
160  LOCATE 25,0:PRINT "PAINT(tiling) demonstration"
170  BTILE$=STRING$(3,CHR$(0))
180 '
190  Y=9
200  FOR BLUE=0 TO 2
210    X=0
220    FOR RED=0 TO 2
230      FOR GREEN=0 TO 2
240        C(0)=BLUE:C(1)=RED:C(2)=GREEN
250        GOSUB *MAKETILE
260        LINE(X,Y)-STEP(69,59),7,B
270        IF C(0)=0 AND C(1)=0 AND C(2)=0 THEN 290
280        PAINT(X+1,Y+1),TILE$,7,BTILE$
290        X=X+70
300      NEXT GREEN
310    NEXT RED
320    Y=Y+60
330  NEXT BLUE
340  END
350 '---  bit set sub.  ---
360 *MAKETILE
370  FOR I=0 TO 2
380    ON C(I) GOTO 400,410
390    TILE(I)=0   :GOTO *EXIT
400    TILE(I)=&HAA:GOTO *EXIT
410    TILE(I)=&HFF
420 *EXIT:NEXT I
430 '---  tile generater  ---
440  TILE$=""
450  FOR I=0 TO 2
460    TILE$=TILE$+CHR$(TILE(I))
470  NEXT I
480  RETURN
```



A.3 ウィンドウとビューポートの使い方

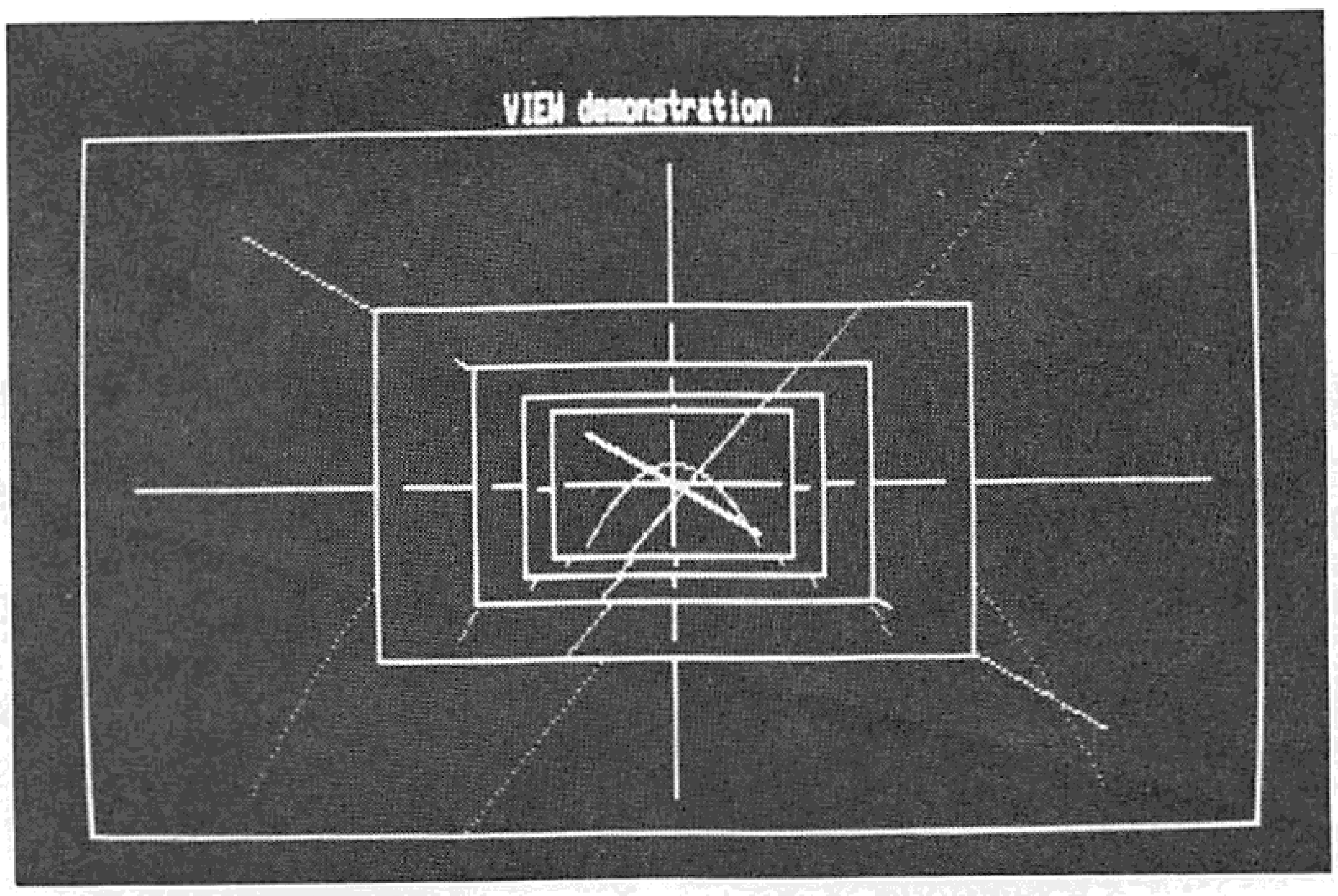
ここではウィンドウとビューポートの概念を具体的に目で見て理解して頂くために，全く同一のグラフを表示するサブルーチンを題材に説明します．このグラフは4象限の直交座標に〝Y＝X″，〝Y＝－2＊X＋15″，〝Y＝X＊X／50－30″の関数をプロットするものです．

VIEW　sample program

このプログラムは，ウィンドウを固定しておき（160行），ビューポートの大きさを変化させながら先程のグラフ表示サブルーチン(270行以降)を呼んでいます．結果はご覧の通りで，グラフそのものの形状及び情報はそのままで，表示される領域の大きさだけが変化していきます．

```
100 '--  VIEW sample program  --
110 '
120  DEFINT A-Z
130  WIDTH 80,25:CONSOLE 0,25,0,1
140  SCREEN 0,0:COLOR 7,0:CLS 3
150  LOCATE 29,0:PRINT "VIEW demonstration"
160  WINDOW(-110,-110)-(110,110)
170 '
180  FOR I=1 TO 5
190    VIEW(320-310/I,100-90/I)-(320+310/I,100+90/I),0,3
200    GOSUB *GRAPH
210    FOR J=0 TO 2000:NEXT J
220  NEXT I
230  SCREEN 0
240  END
250 '-- graph display routine --
260 *GRAPH
270  LINE(-100,0)-(100,0),7
280  LINE(0,-100)-(0,100),7
290  FOR X=-80 TO 80
```

```
300     PSET(X,X),1              ' Y=X
310     PSET(X,-2*X+30),2        ' Y=-2*X+15
320     PSET(X,X*X/50-30),4      ' Y=X*X/50-30
330   NEXT X
340   RETURN
```



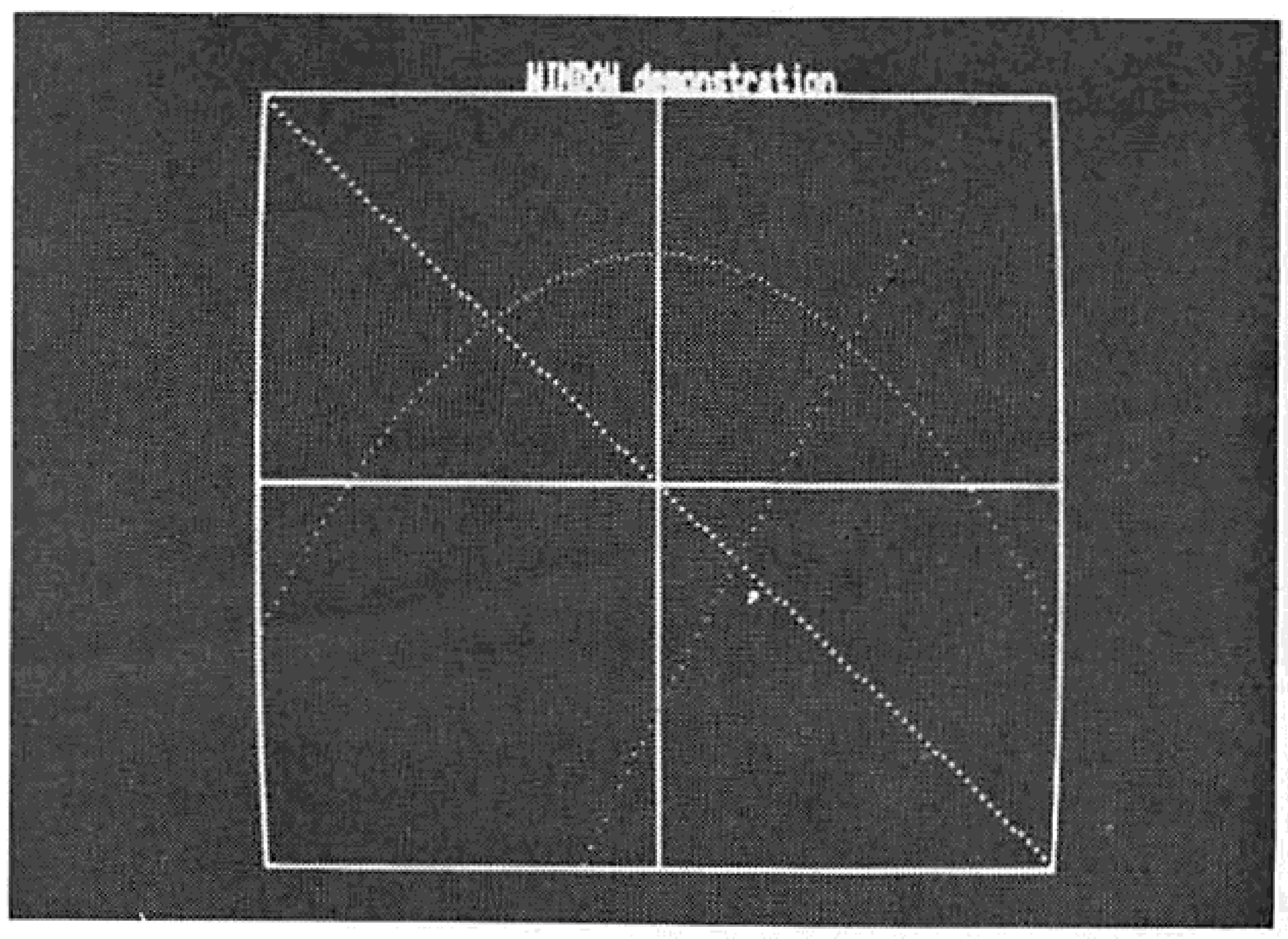
WINDOW　sample program

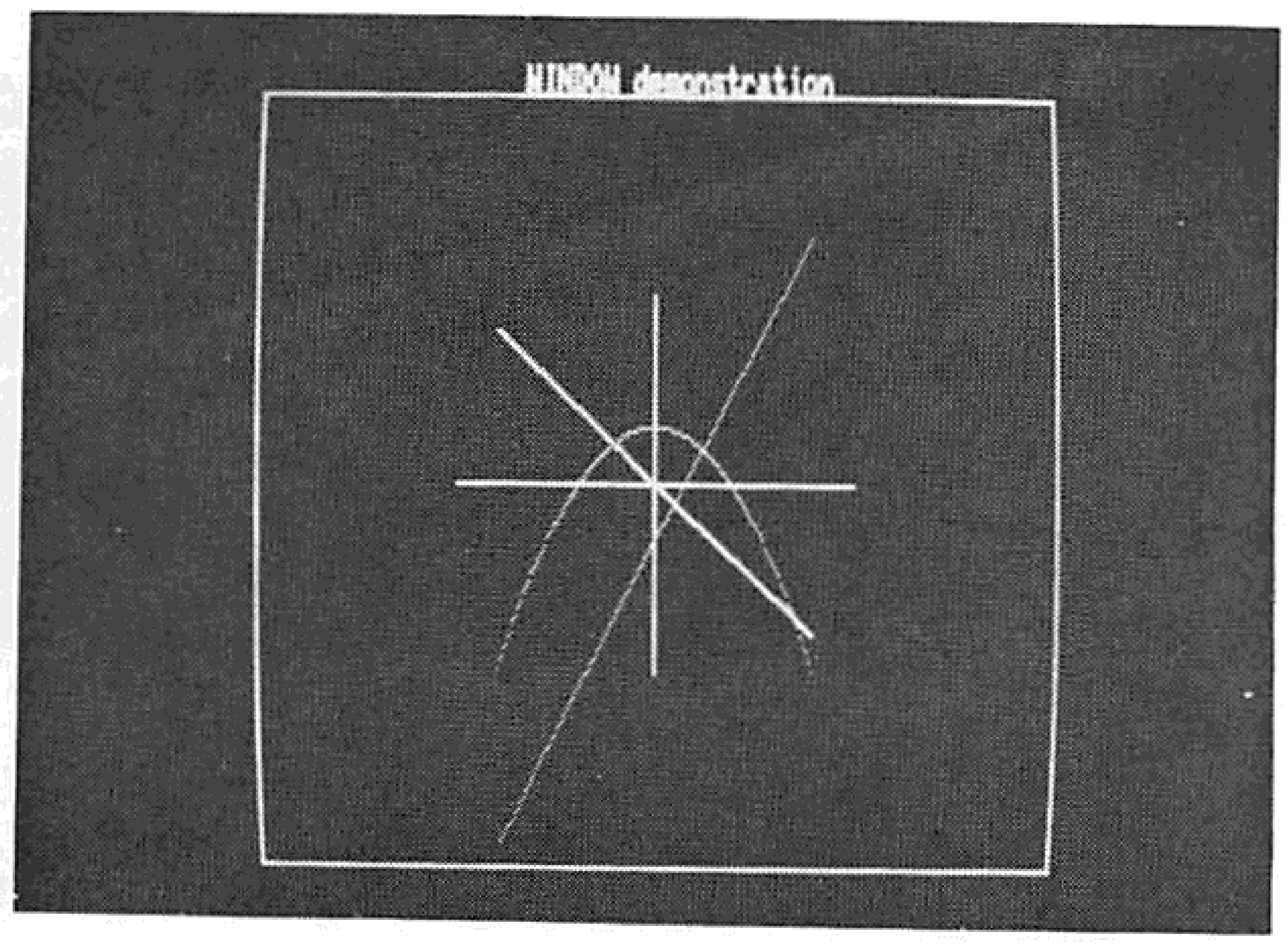
このプログラムは"VIEW sample program"の150，160，190行を変更し，215行を1行追加しただけのものです．VIEW の場合とは逆にビューポートを固定しておき（160行），ウィンドウを変化させながらグラフ表示サブルーチンを呼んでいます．その結果，画面上の表示領域の大きさは常に同じですが，表示されるグラフはだんだん拡大されたものになっていきます．

このようにウィンドウとビューポートを適当に変化されることによって同じ表示のサブルーチンであっても，画面上の任意の領域に拡大，縮小した図形が表示できますから，グラフのアレンジ，画面上でのレイアウトなど容易に行うことができます．

```
100 '--  WINDOW sample program  --
110 '
120  DEFINT A-Z
130  WIDTH 80,25:CONSOLE 0,25,0,1
140  SCREEN 0,0:COLOR 7,0:CLS 3
150  LOCATE 29,0:PRINT "WINDOW demonstration"
160  VIEW(100,8)-(500,198),,3
170 '
180  FOR I=1 TO 5
190    WINDOW(-200/I,-200/I)-(200/I,200/I)
200    GOSUB *GRAPH
210    FOR J=0 TO 2000:NEXT J
215    CLS 2
220  NEXT I
230  SCREEN 0
240  END
```

```
250 '-- graph display routine --
260 *GRAPH
270  LINE(-100,0)-(100,0),7
280  LINE(0,-100)-(0,100),7
290  FOR X=-80 TO 80
300    PSET(X,X),1           ' Y=X
310    PSET(X,-2*X+30),2     ' Y=-2*X+15
320    PSET(X,X*X/50-30),4 ' Y=X*X/50-30
330  NEXT X
340  RETURN
```





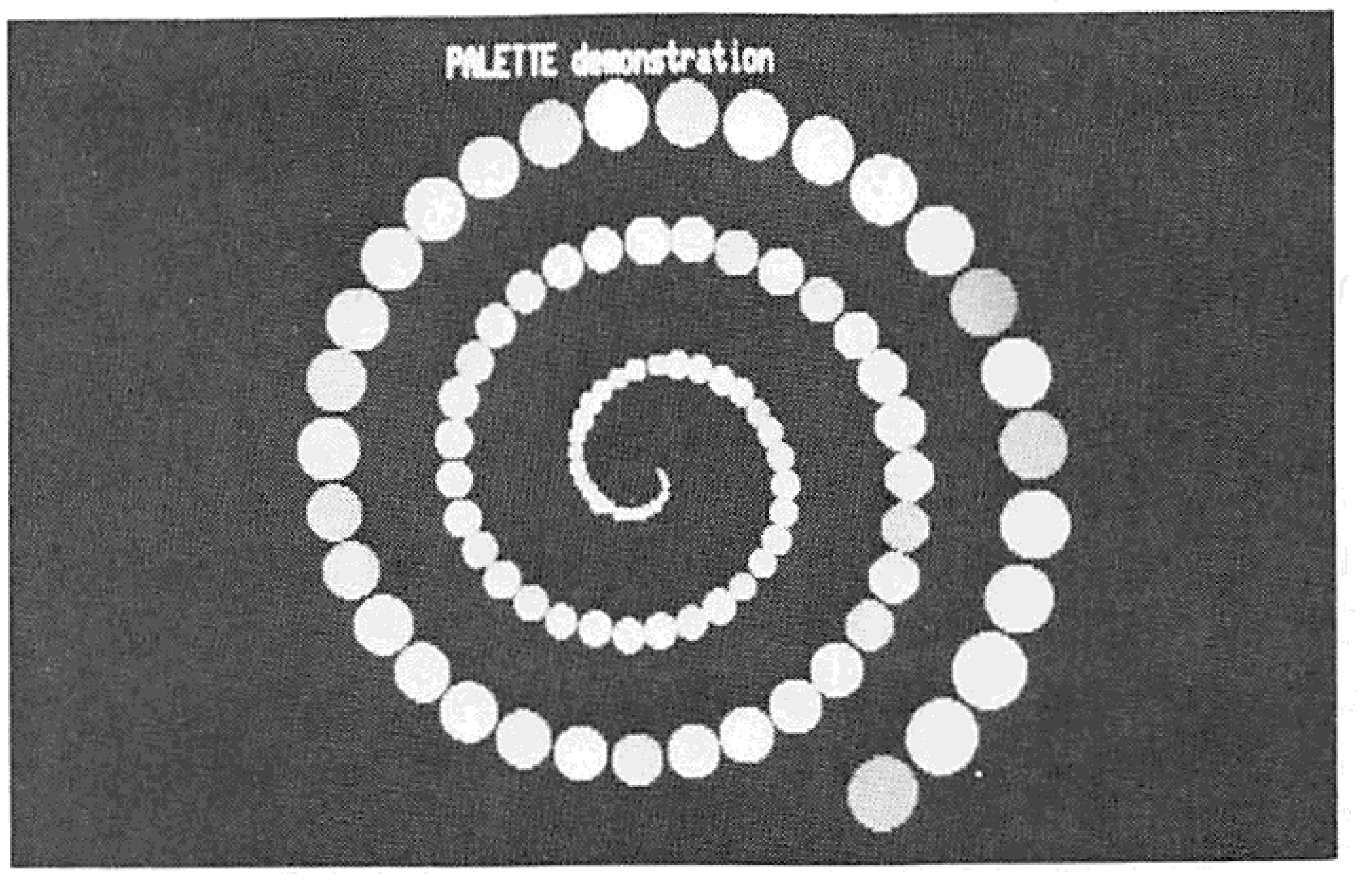
A.4 パレットの使い方

N$_{88}$-BASICのパレット命令をうまく使うと画面上で静止している図形を動いているように見せることもできます．ここでは，最初うず巻き状に配置された円を描いておいて，パレットを順に回転させています．

プログラム上の340行以降が円を表示しているサブルーチンであり，メインプログラムの180行から250行のFORループでパレットを回転させています．＊PINIはパレットを初期状態に戻すサブルーチンです．パレットを変更した場合，必ずこのようにパレットを初期状態に戻すように心がけて下さい．

```
100 '-- PALETTE sample program --
110 '
120  DEFINT A-Z
130  SCREEN 0,0:COLOR 7,0,0,7
140  CONSOLE 0,25,0,1:CLS 3
150  LOCATE 25,0:PRINT "PALETTE demonstration"
160  GOSUB *PINI:GOSUB *CRCL
170 '-- palette rotation --
180  FOR J=1 TO 100
190    CO=1
200      FOR I=1 TO 7
210        COLOR=(I,((CO+I) MOD 7)+1)
220      NEXT
230      CO=CO+1
240    IF CO<6 THEN 200
```

```
250  NEXT
260 '
270  GOSUB *PINI:CLS 3
280  END
290 '-- palette initialize  --
300 *PINI:FOR I=0 TO 7
310     COLOR=(I,I)
320  NEXT:RETURN
330 '-- circle display --
340 *CRCL:DEFSNG R,T:C=1:R=1
350  FOR TH=0 TO 20 STEP .2
360     R=R+1
370     X=COS(TH)*R*2+300:Y=SIN(TH)*R+105
380     CIRCLE(X,Y),R/6,C
390     PAINT(X,Y),C,C
400     C=(C MOD 7)+1
410  NEXT TH:RETURN
```



A.5 割り込みキーの使い方

N_{88}-BASICは3種類の割り込みキーをサポートしています．ここではこれらの3種類の割り込みキー（プログラマブル・ファンクションキー，ヘルプキー，ストップキー）をすべて使ったプログラムを例に，それぞれの役割と使い方を説明します．

このプログラムは0から，入力された数値Nまでの2乗と平方根を求めるもので，各割り込みキーは次のように機能しています．

1) プログラマブル・ファンクションキー（PFキー）

○ PF 1——プリント・オフ．このキーを押すと以後画面への表示をしなくなります．

○ PF 2——プリント・オン．このキーを押すと以後画面に表示します．

140行で処理ルーチンの開始行を定義し，170行でキーをオンしています．

2) ヘルプキー

メッセージの出力．入力の仕方，プログラムの終了の仕方などのメッセー

ジを表示します.

ヘルプキーでINPUT文実行中に割り込んだ場合，単にRETURN命令で復帰させるとINPUT文の次から再開することになり正常に入力されないまま実行に移ってしまいます．そこで，処理ルーチンの中でそのフラグを返して実行に入る前に判定しています（230行）.

3）ストップキー

ストップキー・マスク．プログラム実行中に不注意でストップキーを押して実行が中断されないようにマスクしています.

このようにそれぞれの割り込みキーはその特徴を心得た上で使ってください．割り込みキーを機能的に使うとスマートなプログラムが書けますし，実行速度の向上にもつながります.

```
100 '--  KEY INTERRUPT sample program  --
110 '
120   WIDTH 80:CONSOLE 0,25,0,1
130   COLOR 7,0,0,7:CLS
140   ON KEY  GOSUB *PRINTOFF,*PRINTON
150   ON STOP GOSUB *MASK
160   ON HELP GOSUB *MESSAGE
170   KEY(1) ON:KEY(2) ON
180   STOP ON:HELP ON
190 '
200 *ENTER
210   N=0:FLAG=0
220   INPUT "N =";N
230   IF FLAG<>0 THEN *ENTER
240   IF N<0     THEN *EXIT
250   IF N>100   THEN *ENTER
260   PRINT
270 '
280   HELP OFF:PSW=1
290   FOR I=0 TO N
300     PRINT "N =";I;
310     IF PSW=0 THEN PRINT:GOTO 340
320     PRINT TAB(15);"N*N =";I*I;
330     PRINT TAB(30);"SQR(N) =";SQR(I)
340   NEXT I:PRINT
350   HELP ON:GOTO *ENTER
360 '
370 *EXIT:'-- end routine --
380   KEY OFF:HELP OFF:STOP OFF
390   END
400 *PRINTOFF:'-- print off --
410   PSW=0:RETURN
420 *PRINTON:'-- print on  --
430   PSW=1:RETURN
```

```
440 *MESSAGE:'-- message routine --
450   PRINT:PRINT "<< Instruction >>":PRINT
460   PRINT "N , N*N , SQR(N) ｦ ｹｲｻﾝ ｼﾏｽ｡"
470   PRINT "N ﾊ 0 ｶﾗ 100 ﾏﾃﾞ"
480   PRINT "N ﾆ 0 ﾐﾏﾝﾉ ｶｽﾞｦ ｲﾚﾙﾄ ｵﾜﾘﾏｽ｡"
490   FLAG=1:PRINT:RETURN
500 *MASK:'-- stop key mask --
510   RETURN
```

```
N =?
<< Instruction >>

N , N*N , SQR(N) ｦ ｹｲｻﾝ ｼﾏｽ｡
N ﾊ 0 ｶﾗ 100 ﾏﾃﾞ
N ﾆ 0 ﾐﾏﾝﾉ ｶｽﾞｦ ｲﾚﾙﾄ ｵﾜﾘﾏｽ｡

N =? 6

N = 0             N*N = 0          SQR(N) = 0
N = 1             N*N = 1          SQR(N) = 1
N = 2
N = 3
N = 4
N = 5             N*N = 25         SQR(N) = 2.23607
N = 6             N*N = 36         SQR(N) = 2.44949

N =? -1
```

A.6 ライトペンの使い方

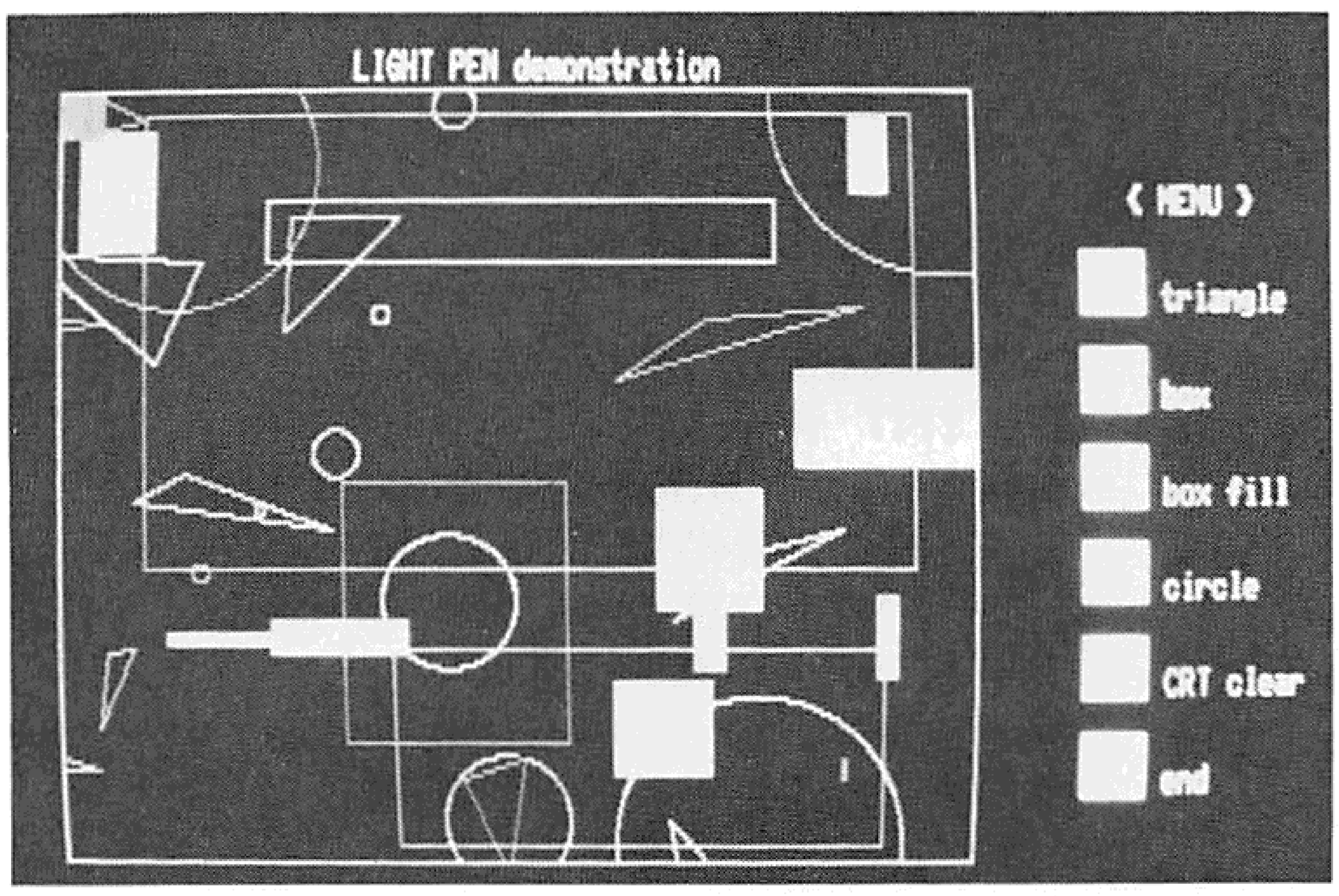
ここで紹介するのはライトペンを使った簡単なグラフィックスのプログラムです．ライトペンを使うと操作性が飛躍的に向上します．

260行から310行でメニューとライトペンでタッチする部分の表示を行っています．ライトペンでタッチする部分は，必ず色が付いていなければなりません．ただし，ライトペンは赤に対する感度が非常に弱いので，赤は使わないようにして下さい．

後はライトペンが押されるたびに処理ルーチンに入りペン関数を読み出してどのメニューが選択されたかを判定しています．

```
100 '--- LIGHT PEN sample program ---
110 '
120  DATA triangle,box,box fill,circle,CRT clear,end
130  SCREEN 0,0:COLOR 7,0,0,7
140  CONSOLE 0,25,0,1:WIDTH 80,25:CLS 3
150  DEFINT A-Z
160  XX=500:YY=300:X=500:Y=300
170  DEF FNX=RND*X:DEF FNY=RND*Y
180  DEF FNC=INT(RND*7+1):DEF FNS=RND*100-50
190  DEF FNR=RND*100
200  P$=CHR$(&H87)+CHR$(&H87):P$=P$+P$
210  ON PEN GOSUB *PENSUB
220  PEN ON
```

```
230 '
240  LOCATE 18,0:PRINT "LIGHT PEN demonstration"
250  LOCATE 66,4:PRINT "< MENU >"
260  FOR I=0 TO 5
270    READ MENU$(I)
280    LOCATE 63,I*3+6:PRINT P$
290    LOCATE 63,I*3+7:PRINT P$
300    LOCATE 68,I*3+7:PRINT MENU$(I)
310  NEXT I
320  VIEW(1,10)-(451,198),,7
330  WINDOW(0,0)-(XX,YY)
340  GOTO 340
350 '
360 *PENSUB
370  PX=PEN(1):PY=PEN(2):PEN OFF
380  IF PX<64 OR PX>67 THEN *EXIT
390  IF  5<PY AND PY<8  THEN GOSUB *TRIANGLE:GOTO *EXIT
400  IF  8<PY AND PY<11 THEN GOSUB *BOX     :GOTO *EXIT
410  IF 11<PY AND PY<14 THEN GOSUB *BOXFILL :GOTO *EXIT
420  IF 14<PY AND PY<17 THEN GOSUB *CIRCL   :GOTO *EXIT
430  IF 17<PY AND PY<20 THEN GOSUB *CRTCLS  :GOTO *EXIT
440  IF 20<PY AND PY<23 THEN GOSUB *ENDER
450 *EXIT:PEN ON:RETURN
460 '
470 *TRIANGLE
480  SX=FNX:SY=FNY:C=FNC
490  LINE (SX,SY)-STEP(FNS*2,FNS),C
500  LINE -STEP(FNS*2,FNS),C
510  LINE -(SX,SY),C
520  RETURN
530 *BOX
540  LINE(FNX,FNY)-(FNX,FNY),FNC,B
550  RETURN
560 *BOXFILL
570  LINE (FNX,FNY)-STEP(FNS*2,FNS),FNC,BF
580  RETURN
590 *CIRCL
600  CIRCLE (FNX,FNY),FNR,FNC
610  RETURN
620 *CRTCLS
630  CLS 2:RETURN
640 *ENDER
650  PEN OFF:END
```



A.7 RS-232Cの使い方

RS-232C では，通信機能として，ターミナルモードにより外部のコンピュータのターミナルとして使用するものと，BASIC モードにより RS-232C を一つのファイルとして扱い，BASIC の管理下において使用するものの，二つがあります．

ここでは，BASIC モードによる使い方を説明します．

このプログラムは，PC-8801MKII 2 台をケーブルで接続し，データを送受信するものです．

まず，送信側では，始めに通信形式を設定します．各パラメータは TERM文と同じです．次に文字変数に送信するデータを文字として与えます． 120行で送信を行います．ファイル番号はOPEN 文で指定したファイル番号です．最後にファイルをCLOSEします．

一方，受信側では送信側と同じ通信形式を設定し，モードをINPUTにします．110行で送信されてきたデータを受信し,120行でそのデータを画面に表示します．そしてファイルをCLOSEします．

送信側

```
100  OPEN "COM : E73XS" FOR OUTPUT AS #1
110  A$= "NEC PC-8801MK2"
120  PRINT  #1, A$
130  CLOSE
```

受信側

```
100  OPEN "COM : E73XS" FOR INPUT AS #1
110  INPUT  #1, A$
120  PRINT  A$
130  CLOSE
```

次に，RS-232C による割り込みを説明します．ON COM GOSUB 文で割り込みが起った場合の処理ルーチンを定義します．COM ON で割り込みを許可します．これ以後，RS-232C から入力割り込みがあると，定義されている処理ルーチンへ制御を移します． RETURN 文で中断した所から再開します．

このプログラムは，送信，受信の両方に使用でき，割り込みによって受信されるようになっています．割り込みがない場合は，160～180 行の処理を繰り返し，キーボードからは 1 文字ずつ入力されるので，その文字をデータとして 1 文字ずつ送信し，また，入力がない場合も 160～180 行の処理を繰り返しますがなにも送信は

しません．また，この間，割り込みが発生すると，RECEVCHAR のルーチンに飛んで，送信されてくる文字を画面に表示し，RETURN 文で戻り160～180行の処理を繰り返します．

```
100 FALSE=0
110 TRUE=NOT FALSE
120 ECHO=TRUE
130 OPEN "COM:E73XS" AS #1
140 ON COM GOSUB *RECEVCHAR
150 COM ON
160 C$=INKEY$
170 IF C$<>"" THEN PRINT #1,C$;:IF ECHO THEN PRINT C$;
180 GOTO 160
190 *RECEVCHAR
200 IF LOC(1)=0 THEN RETURN
210 N$=INPUT$(LOC(1),#1)
220 PRINT N$;
230 RETURN
```

B　周辺装置との入出力（N88-BASIC）

PC-8801MKIIは1章のファイルディスクリプタで説明した通り多くの周辺機器との入出力が可能であり，入出力の仕方もファイルという概念で取り扱うために，汎用性をもっており同じ命令がどのデバイスに対しても有効です（例外はある.）

ここでファイルを扱うための基本的な命令について，どのデバイスに対して有効なのかを表にしておきます.

デバイス／入出力命令	ディスクn：	CASn：	COM：	KYBD：	SCRN：	LPT：	備　考
BLOAD	○	×	○	○	×	×	○：使用できる. ×：使用できない. ＊：実際には何も行われない. SCRN：とKYBD：とディスクn：はディスク版のみ可
BSAVE	○	×	○	×	○	○	
CLOSE	○	○*	○	○*	○*	○	
EOF	○	×	○	×	×	×	
FPOS	○	×	×	×	×	○	
GET	○	○	×	○	×	×	
INPUT#	○	○	○	○	×	×	
INPUT$	○	×	○	○	×	×	
LINE INPUT#	○	○	○	○	×	×	
LOAD	○	○	○	○	×	×	
LOC	○	×	○	○	×	×	
LOF	○	×	○	×	×	×	
OPEN	○	○*	○	○*	○*	○	
PRINT# (PRINT# USING WRITE#)	○	○	○	×	○	○	
PUT	○	○	×	×	○	○	
SAVE	○	○	○	×	○	○	
WIDTH	×	×	○	×	×	○	

各命令の意味は2章，3章を参照してください．次に各デバイスで固有の事柄について説明します.

B.1 プリンタ

PC-8801MKII で接続可能なプリンタは1系統で，ファイルディスクリプタは〝LPT1：〃で表されます．単に〝LPT：〃と，1を省略してもかまいません．

プリンタは最も頻繁に使用するデバイスの1つですから，一般の出力命令の他に専用の命令も数多く用意されています．これらは特に，〝L〃で始まる命令に統一されています．次にそれらの命令を列挙します．

LPRINT, LPRINT USING
LFILES, LLIST, WIDTH LPRINT
LPOS, COPY

B.2 カセット

N_{88}-BASICでは補助記憶装置としてカセットが用いられるよう配慮されています．カセットファイルを操作する場合には装置名として〝CAS1：〃または〝CAS2：〃を指定します．前者は1200ボー，後者は600ボーの転送速度でカセット入出力を行うことを意味します．他の入出力装置名と異なり1と2は同一のカセットに対し転送速度を指定するのであって，2台のカセットが使えるわけではないことに注意してください．1200ボーの方が高速なため同一のプログラム，データでも短時間で入出力することができます．

LOADコマンドによりカセットからプログラムをロードする際には，BASICはカセットファイルを見つけるごとにそのファイル名を表示します．見つけたファイルがLOADで指定されたファイルと異なる場合には，

Skip：〈ファイル名〉

と表示し，指定されたファイル名の探索を続行します．指定されたファイルが見つかるとBASICは

Found：〈ファイル名〉

と表示し，メモリへのロードを開始するとともに，画面の右上隅にロード中であることを示すためにアスタリスク（＊）を点滅させます．

ロード中には，ストップキーあるいはCTRL-Cによりロードを中止することができます．またその時に，まだ指定されたファイルを見つけていない場合にはメモリ中に入っていたプログラムは保存されます．

SAVEも同様に行えます．ただし，ディスクのようにAオプションを付けてア

スキー形式でセーブすることはできません（Aオプションの指定は無視されます）．

また，SAVE後，正しくプログラムがセーブされたかをLOAD？により確認することができます．

LOAD? "CAS1 : XYZ"

この例では，カセット上にプログラムファイル"XYZ"が1200ボーで正しくセーブされたかを，メモリ中のプログラムと比較，確認します．

またBASICからカセットレコーダのモータを制御することもできます．これはMOTOR命令によって行うことができますが，LOAD, SAVEの実行では自動的に管理されます．

B.3 ディスク

フロッピーディスクは現在パーソナルコンピュータの外部記憶装置として最も汎用的なデバイスです． N88DISK-BASICはこれを操作する専用の命令を数多く備えています．

B.3.1 プログラムファイルコマンド

1) CHAIN

指定したプログラムをディスクからロードし，メモリ上のプログラムとチェインします（詳しくは2章CHAIN参照）．

2) MERGE

プログラムファイルをディスクからメモリ上にロードしますが，現在のメモリ内容の消去は行いません．すなわち，メモリ上のプログラムとディスク上のプログラムを結合します．もし同一の行番号が存在した場合，ディスク上のプログラム行が残ります．マージのためのプログラムファイルはアスキー形式でなければなりません．MERGEコマンド実行後はコマンドレベルになります．

3) KILL

ディスク上のファイルを削除します．すべてのディスクファイルに対して使用できますが，ファイルは閉じられていなければなりません．

4) NAME

ディスクファイルの名前を変更します．すべてのディスクファイルに対して使用できます．

5) DSKO$

通常のファイル操作とは無関係に，ディスケット上の任意の位置にデータを書き込みます．

6) DSKI$

通常のファイル操作とは無関係に，ディスケット上の任意の位置のデータを読み出します．これはディスクの内容をダンプするには便利です．

※但しDSKO$とDSKI$はディスクドライブの種類によって，ダイナミックに文法が変化しますから気を付けてください．

7) DSKF

接続されているディスクドライブの諸元を与えます．DSKO$, DSKI$を実行する場合，この命令によって対象となるディスクドライブの諸元（片面か両面か，トラック，セクタ）を先に調べておく必要があります．

B.4 RS-232C

PC-8801MKIIは外部との通信機能としてRS-232Cインターフェイスを内蔵しています．また，N_{88}-BASICでは，これを制御する命令としてOPEN, CLOSE, PRINT #, INPUT#などが用意されており，ON COM GOSUB, COM ON／OFF／STOP文により割り込みでの制御も可能です．

さらにTERMコマンドにより，このRS-232C回線を通してPC-8801MKIIを外部のコンピュータのターミナルとして使用することもできます． その際N_{88}-BASICが持つ機能のいくつかは，リモートBASICプロトコルにより使用可能となります．

これらの詳しい使用法はユーザーズ・マニュアルを参照してください．

C　データファイルの作り方

データファイルには，シーケンシャルファイルとランダムファイルの２種類があります．ここでは入出力の対象としてディスクを例にとりこの２つのファイルの作り方を説明します．カセットを対象とする場合は，ランダムファイルを作ることはできませんが，シーケンシャルファイルは同様な方法で作ることができます．

C.1　シーケンシャルファイル

シーケンシャルファイルは，ランダムファイルよりも簡単に作ることができますが，データをアクセスする時に，融通性とスピードの点でランダムファイルより制限を受けます．これは，シーケンシャルファイルがデータを順番に記録し，読み出す時も最初から順番に読まなければならないからです．

シーケンシャルファイルをアクセスする際に使用されるステートメントと関数を以下に示します．

OPEN, CLOSE, INPUT#, LINE INPUT#, PRINT#, PRINT# USING, WRITE#, EOF, LOF, LOC, INPUT$ (X, 〔#〕 Y)

シーケンシャルファイルを作り，その中のデータをアクセスするために必要な手順は次のようになります．

C.1.1　シーケンシャルファイルの作成

1)　OUTPUT モードでファイルをオープンします．

```
OPEN "TEST" FOR OUTPUT AS #1
```

2)　WRITE #，PRINT # 等を使ってファイルにデータを書き込みます．

```
WRITE #1, N$ ; P
```

3)　必要なだけ2)の作業を繰り返した後，ファイルをクローズして終了します．

```
CLOSE #1
```

例として，キーボードから生徒のテストの点数を入力して，シーケンシャルファイル "TEST" を作るプログラムを示します．（プログラム C - 1)

```
100 REM ﾌﾟﾛｸﾞﾗﾑ C-1
110  OPEN "TEST" FOR OUTPUT AS #1
120 *LOOP
130  INPUT "ﾅﾏｴ ";N$
140  IF N$="end" THEN *FIN
150  INPUT "ｶﾓｸ ";K$
160  INPUT "ﾄｸﾃﾝ";P
170  WRITE #1,N$;K$;P
180  PRINT
190 GOTO *LOOP
200 *FIN
210  CLOSE #1:END
Ok
run
ﾅﾏｴ ｱｵｷ ｼｹﾞﾙ
ｶﾓｸ ﾘｶ
ﾄｸﾃﾝ 68

ﾅﾏｴ ﾀｸﾞﾁ ﾖｼｵ
ｶﾓｸ ﾘｶ
ﾄｸﾃﾝ 49

ﾅﾏｴ ﾋｭｳｶﾞ ﾚｲｺ
ｶﾓｸ ﾘｶ
ﾄｸﾃﾝ 85

ﾅﾏｴ ﾜﾀﾅﾍﾞ ｼﾞｭﾝ
ｶﾓｸ ﾘｶ
ﾄｸﾃﾝ 45

ﾅﾏｴ end
Ok
```

N - BASIC ではWRITE #文がありませんから，

PRINT #1, N$; 〝,〞 ; K$; 〝,〞 ; P

のようにデータを正しく区切って書き込まなければなりません.

シーケンシャルファイルを作成するプログラムにおいて，PRINT # USING を使用することによって，書式制御したデータを書き込むこともできます．例えば，

PRINT #1, USING 〝####. ##,〞 ; A,B,C,D

を使うと，特に区切り記号を付けることなく，ディスクに数値データを書き込むことができます．この区切り記号とは，シーケンシャルファイル中のデータの読み込みにINPUT# を使用するとき，数値データの場合はコンマ，キャリッジリターン，ラインフィード，空白などを区切りとしてデータを読み込むためのものです．LINE INPUT#を使用する場合は，キャリッジリターンのみを区り記号とします．また，PRINT# の代わりにWRITE# を使用しますと，区切り記号を強制的

に出力するため，ディスクの使用領域を節約できます．

LOC関数をシーケンシャルファイルについて使用する場合は，そのファイルが開かれて以降，アクセスされた総セクタ数を値とします．

C.1.2 シーケンシャルファイルからの読み込み

1) INPUTモードでファイルをオープンします．

   OPEN "TEST" FOR INPUT AS #1

2) INPUT #，LINE INPUT#等を使ってファイルからデータを読み込みます．

   INPUT #1，N$，P

3) 必要なだけ2)の作業を繰り返した後，ファイルをクローズして終了します．

   CLOSE #1

次にプログラムC-2として，プログラムC-1で作ったファイル "TEST" をアクセスし，50点以下の生徒を全部表示する例を示します．

```
100 REM ﾌﾟﾛｸﾞﾗﾑ C-2
110  OPEN "TEST" FOR INPUT AS #1
120 *LOOP
130  INPUT #1,N$,K$,P
140  IF P>=50 THEN *LOOP
150  PRINT N$,K$,P
160 GOTO *LOOP
Ok
run
ﾀｸﾞﾁ ﾖｼｵ        ﾘｶ             49
ﾜﾀﾅﾍﾞ ｼﾞｭﾝ      ﾘｶ             45
Input past end in 130
Ok
```

プログラムC-2ではファイル中のデータを最初から順番に読み込んでいきます．そして，ファイルの最後でもう読み込むデータがなくなると，130行で"Input past end" エラーが起こります．これを避けるためには，125行に

```
125 IF EOF(1) THEN *FIN
```

を挿入してファイルの終りをチェックし

```
170 *FIN
180  CLOSE #1
190 END
```

のように処理します.

C.1.3 シーケンシャルファイルへのデータの追加

ディスク上にシーケシャルファイルが既に作られていて，その後にデータを追加したい場合には，次のような手順で行います.

1) APPEND モードでファイルをオープンします.

```
OPEN "TEST" FOR APPEND AS #1
```

2) WRITE#, PRINT# 等を使ってファイルに追加したいデータを書き込みます.

```
WRITE #1, N$ ; P
```

3) 必要なだけ2)の作業を繰り返した後,ファイルをクローズして終了します.

```
CLOSE #1
```

例として，プログラム C-1 で作ったファイル "TEST" にデータを追加する例をプログラム C-3 として示します.

```
100 REM ﾌﾟﾛｸﾞﾗﾑ C-3
110  OPEN "TEST" FOR APPEND AS #1
120 *LOOP
130  INPUT "ﾅﾏｴ ";N$
140  IF N$="end" THEN *FIN
150  INPUT "ｶﾓｸ ";K$
160  INPUT "ﾄｸﾃﾝ";P
170  WRITE #1,N$;K$;P
180  PRINT
190 GOTO *LOOP
200 *FIN
210  CLOSE #1:END
Ok
run
ﾅﾏｴ ｱｵｷ ｼｹﾞﾙ
ｶﾓｸ ｺｸｺﾞ
ﾄｸﾃﾝ 40
ﾅﾏｴ ﾀｸﾞﾁ ﾖｼｵ
ｶﾓｸ ｺｸｺﾞ
ﾄｸﾃﾝ 75
```

```
ﾅﾏｴ ﾋｭｳｶﾞ ﾚｲｺ
ｶﾓｸ ｺｸｺﾞ
ﾄｸﾃﾝ 80
ﾅﾏｴ ﾜﾀﾅﾍﾞ ｼﾞｭﾝ
ｶﾓｸ ｺｸｺﾞ
ﾄｸﾃﾝ 60
ﾅﾏｴ end
Ok
```

このプログラムはプログラムC-1の110行をAPPENDモードに替えただけです．従って，このプログラムでデータを追加したファイルに対してプログラムC-2を実行してチェックすることができます．

```
run
ﾀｸﾞﾁ ﾖｼｵ        ﾘｶ             49
ﾜﾀﾅﾍﾞ ｼﾞｭﾝ      ﾘｶ             45
ｱｵｷ ｼｹﾞﾙ        ｺｸｺﾞ           40
Ok
```

C.1.4 シーケンシャルファイルのデータの修正

シーケンシャルファイルではそのファイルのデータを修正することができません．従って，現在のファイルから読み込んで修正したデータを新しい別のシーケンシャルファイルへ書き込んでいく作業を行わなければなりません．以下にプログラムC-4として，プログラムC-1やC-3で作ったファイルの修正を行う例を示します．

```
100 REM ﾌﾟﾛｸﾞﾗﾑ C-4
110  OPEN "TEST" FOR INPUT AS #1
120  OPEN "test" FOR OUTPUT AS #2
130 *LOOP
140  IF EOF(1) THEN *FIN
150  INPUT #1,N$,K$,P
160  IF FLG=1 THEN *WR
170  PRINT N$,K$,P
180 *QUESTION
190  PRINT "ﾍﾝｺｳ ｶﾞ ｱﾘﾏｽｶ ";
200  INPUT "(y or n or d or e )";A$
210  A$=LEFT$(A$,1)
220  IF A$="e" THEN FLG=1:GOTO *WR
230  IF A$="d" THEN *LOOP
240  IF A$="n" THEN PRINT:GOTO *WR
250  IF A$<>"y" THEN *QUESTION
260  INPUT "ﾅﾏｴ ";N$
270  INPUT "ｶﾓｸ ";K$
280  INPUT "ﾄｸﾃﾝ";P
290  PRINT
300 *WR
310  WRITE #2,N$;K$;P
320 GOTO *LOOP
```

```
330 *FIN
340  CLOSE #1
350  CLOSE #2
360  KILL "TEST"
370  NAME "test" AS "TEST"
380 END
OK
```

上のプログラムでは，ファイル〝TEST″から読み込んだデータに修正がない場合はそのまま新しいファイル〝test″へ書き込みます．もし修正がある場合は，260～280行で新しいデータを入力し〝test″へ書き込みます．削除する場合には書き込まずに次のデータを〝TEST″から読み込みます．修正が終了した場合には，〝TEST″から読み込んだデータをそのまま〝test″へ書き込む作業を〝TEST″のファイルエンドまで繰り返します．

最後に，古いファイル〝TEST″を消去し，〝test″を〝TEST″に名前を変更して終了します．

C.2 ランダムファイル

ランダムファイルを作るには，シーケンシャルファイルの場合より多くの手順が必要となります．

しかし，ランダムファイルでは数値を内部表現に変換して記録するためファイルの占める領域を小さくすることができます．また，レコード単位で任意にデータのアクセスができるため，シーケンシャルファイルの場合のように空読みする必要がありません．さらに，データの修正を自由に行えるという利点もあります．

ランダムファイルを作ったりアクセスしたりするには次の手順が必要となります．

1) ランダムファイルをオープンします．

   OPEN 〝TESTR″ AS #1

2) ランダムファイルバッファに文字変数を割り当てます．

   FIELD #1, 20 AS N$, 20 AS K$, 2 AS P$

3) データの書き込み，読み込みを行います．

   PUT #1, NUM%

GET #1, NUM%

ただし，データを書き込む前に，バッファへデータを転送しておかなければなりません．データが文字列の場合は LSET，RSET を使います．

LEST N$=A$

また，数値である場合には一旦 MKI$，MKS$，MKD$を使って，それぞれ整数値，単精度実数値，倍精度実数値を 2 バイト，4 バイト，8 バイトの文字に変換してからバッファへ転送します．

LSET P$=MKI$ (P)

また，GET# によって読み込んだデータが数値である場合には，CVI，CVS，CVD を使って文字列から数値へ変換します．

P=CVI (P$)

4) ランダムファイルをクローズして作業を終了します．
CLOSE #1

次に，プログラム C-5 として，プログラム C-1～C-4 を生徒番号によってランダムアクセスを行うランダムファイルにした場合の例を示します．

```
100 REM ﾌﾟﾛｸﾞﾗﾑ C-5
110  INPUT "ﾌｧｲﾙ ﾒｲ ";F$
120  OPEN F$ AS #1
130  FIELD #1,1 AS ID$,20 AS N$,20 AS K$,2 AS P$
140 *FUNC
150  CLS
160  PRINT "   1:ﾃﾞｰﾀ ｻｸｾｲ,ﾍﾝｺｳ"
170  PRINT "   2:ﾃﾞｰﾀ ｻﾝｼｮｳ"
180  PRINT "   3:ﾃﾝｽｳ ﾆ ﾖﾙ ｹﾝｻｸ"
190  PRINT "   4:ｼｭｳﾘｮｳ"
200  PRINT "999:ﾌｧｲﾙ ﾉ ｲﾆｼｬﾗｲｽﾞ"
210  INPUT "   ｴﾗﾝﾃﾞｸﾀﾞｻｲ ";F
220   IF F=999 THEN *FINI
230  ON F GOTO *HENKO,*SANSHO,*KENSAKU,*FIN
240 GOTO *FUNC
250 *SANSHO
260  SFLG=-1
270 GOTO *HKLOOP
280 *HENKO
290  SFLG=0
300 *HKLOOP
310  GET #1,1
320  MAX=CVI(P$)
330  PRINT "ｾｲﾄ ﾊﾞﾝｺﾞｳ (1-";USING "###";MAX-1;
```

```
340  INPUT " or 0:end)";NUM%
350   IF NUM%=0 THEN *FUNC
360  NUM%=NUM%+1
370  GET #1,NUM%
380   IF ID$=CHR$(255) AND SFLG=0 THEN *QUESTION
390   IF ID$=CHR$(255) AND SFLG THEN *HKLOOP
400  P%=CVI(P$)
410  PRINT
420  PRINT "ﾅﾏｴ :";N$
430  PRINT "ｶﾓｸ :";K$
440  PRINT "ﾄｸﾃﾝ:";P%
450  PRINT
460   IF SFLG THEN *HKLOOP
470 *QUESTION
480  INPUT "ﾅﾏｴ ";A$
490   IF A$<>"" THEN LSET N$=A$
500  INPUT "ｶﾓｸ ";A$
510   IF A$<>"" THEN LSET K$=A$
520  INPUT "ﾄｸﾃﾝ";A$
530   IF A$<>"" THEN LSET P$=MKI$(VAL(A$))
540  PRINT
550  LSET ID$="U"
560  PUT #1,NUM%
570 GOTO *HENKO
580 *KENSAKU
590  GET#1,1
600  MAX=CVI(P$)
610  INPUT "ﾅﾝﾃﾝｲｶ ﾃﾞ ｹﾝｻｸ ｼﾏｽｶ ";AKA
620  PRINT
630  FOR NUM%=2 TO MAX
640   GET #1,NUM%
650    IF ID$=CHR$(255) THEN *KLOOP
660    IF CVI(P$)>AKA THEN *KLOOP
670   PRINT USING "### ";NUM%-1;
680 *KLOOP:PRINT N$;K$;USING ' ####';CVI(P$)
690  NEXT NUM%
700  PRINT
710  PRINT "ｹﾝｻｸ ｼｭｳﾘｮｳ. ｷｰ ｦ ﾀﾀｲﾃ ｸﾀﾞｻｲ"
720 *LOOP
730  A$=INKEY$:IF A$="" THEN *LOOP
740 GOTO *FUNC
750 *FINI
760  INPUT "ｻｲﾀﾞｲ ﾉ ｾｲﾄ ﾊﾞﾝｺｳ ﾊ ";MAX
770  INPUT "ﾌｧｲﾙ ｦ ｲﾆｼｬﾗｲｽﾞ ｼﾏｽ (yes or no) ";A$
780   IF A$<>"yes" THEN *FUNC
790  FOR NUM%=1 TO MAX+1
800   LSET ID$=CHR$(255)
810   PUT #1,NUM%
820  NEXT NUM%
830  LSET P$=MKI$(MAX+1)
840  PUT #1,1
850 GOTO *FUNC
860 *FIN
870  CLOSE #1
880 END
```

D　機械語プログラムの作り方

N_{88}-BASIC には，機械語で書かれたプログラムを呼び出す機能として，USR 関数と CALL 文が用意されています．ここでは，機械語プログラムを作成する際の注意と USR 関数，CALL 文との引数の受け渡し方について説明します．なお，ここでの説明を正しく理解するためには機械語プログラムに関する知識を必要とします．

D.1　メモリの配置

機械語ルーチンをモニタにより作成したり，ファイルからロードする際には，そのためのメモリ領域を確保する必要があります．この機械語プログラム用のメモリ領域は BASIC の使用する領域と重複してはなりません．さもないと，BASIC の動作が異常になったり，逆に BASIC の動作により準備した機械語プログラムが破壊されたりします．メモリ領域の確保は CLEAR 文により BASIC の使用するメモリの上限を設定することにより行います．指定された上限以上の領域は BASIC は利用しなくなりますので機械語プログラム領域として用いることができます（CLEAR 文参照）．

D.2　機械語プログラムの準備

確保した領域に機械語プログラムを準備する主な方法としては次の 3 つがあります．

1. モニタを用いる
   MON コマンドにより機械語モニタに移り，モニタの機能を利用してプログラムを書き込む（ユーザーズ・マニュアル参照）．
2. BLOAD によりロードする
   既に BSAVE によりセーブされている機械語プログラムをロードする
3. POKE によりプログラムを書き込む．
   比較的短いプログラムであれば，DATA 文により機械語プログラムをデータの型で表現しておき，それを READ 文で読み出した後 POKE 文により書き込む

また機械語プログラムを作成する際には次の点に注意してください．

1. BASIC に制御を戻すためには，プログラムはリターン命令（RET）を用いなければなりません．
2. 機械語ルーチンに引き渡された引数が文字列であった場合には，引数の値は

次のような3バイトからなるストリング・ディスクリプタと呼ばれる領域のアドレスとなります．

(a) ストリング・ディスクリプタの最初のバイトはその文字列の長さ（0から255）を保持します．

(b) 2番目のバイトは文字列の置かれている先頭アドレスの下位8ビットを保持します．

(c) 最後のバイトは，文字列の置かれている先頭アドレスの上位8ビットを保持します．

機械語ルーチンではこの3バイトのストリング・ディスクリプタを書き代えてはなりません．またストリング・ディクリプタによって指されるストリングの内容は変更することはできますが，その長さを変更してはなりません．

3. 機械語ルーチンからBASICに戻る時には，呼び出された時とスタックポインタ（SP）の値が等しくなっていなければなりません．

4. 機械語ルーチンが呼び出された時，スタック領域としては8レベル（16バイト）分だけ利用できるように設定されています．もし，機械語ルーチン内でより大きなスタックを必要とする場合には，ルーチン内で独自にスタックを用意しなければなりません．また，この時，BASICに戻る時にスタック・ポインタをもとに戻すことを忘れてはなりません．

D.3 USR関数の呼び出し

USR関数は

USR〔〈番号〉〕（〈引数〉）

という書式によって呼び出されます．ただし〈番号〉は0から9までの数値で，引数は任意の数値式または文字式です．USR関数は書式上〈引数〉を必要としますので，呼び出されたルーチンが引数を必要としない場合でも，ダミーの引数を与える必要があります．

USR関数が呼び出され，対応する機械語ルーチンに制御が移った時，Aレジスタは，渡された引数の型を示す値を持っています．その値と意味は次のようになっています．

Aの値	引数の型
2	2バイトの整数（2の補数表現）
3	文字列型
4	単精度型

8　倍精度型

引数が文字列の場合には〔DE〕レジスタペアが3バイトから成るストリング・ディスクリプタを指しています（前節参照）．

引数が数値の場合には，〔HL〕レジスタに浮動小数点アキュームレータ(FAC)と呼ばれる8バイトの領域の5バイト目のアドレスが入っています．実際の引数は，このFACの内に各型に応じて以下のように格納されて渡されます．

●整数型のとき

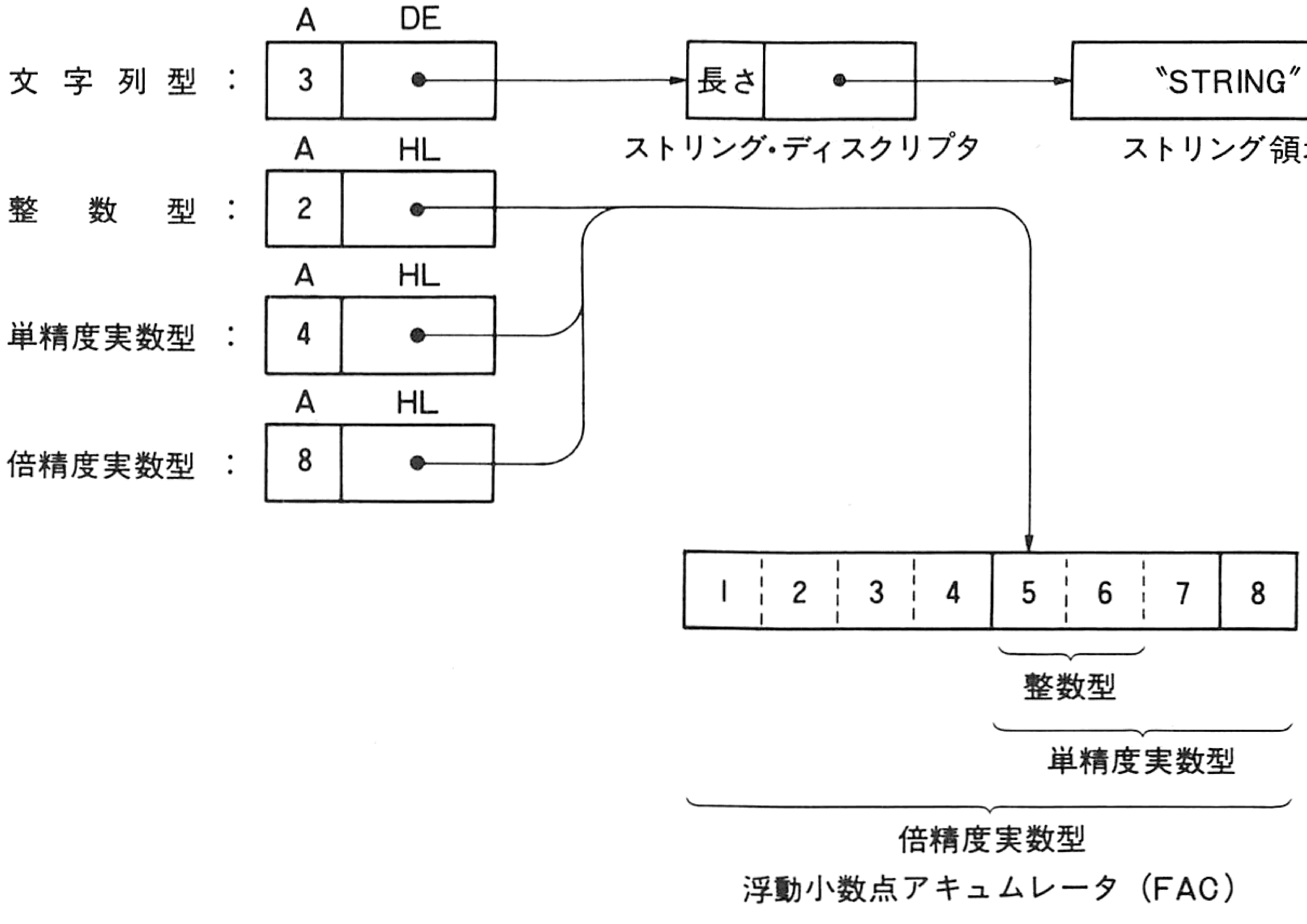
FACの5バイト目（〔HL〕レジスタペアが指しているところ）に引数の下位8ビット，6バイト目に上位8ビットが入ります．

●単精度実数型のとき

FACの8バイト目は指数部になっており，(指数−128)の値が入ります．小数点は仮数部の最上位ビットの左にあると想定します．7バイト目は仮数部の最高部7ビットを保持します．このバイトの最上位ビットは付号を示し，0で正，1で負を表します．6バイト目と5バイト目は，それぞれ仮数部の中部と最低部の8ビットを保持します．

●倍精度実数型のとき

5バイト目から8バイト目までは単精度実数型と同じように指数部と仮数部の上位3バイトが入ります．1バイト目から4バイト目には仮数部の下位4バイトが入ります．



USR関数がBASICに値を返すにはFACで求めた関数値を設定してやることで行えます．通常返す値は，引数として渡した値と同じ型となります．

サンプルプログラム

USR関数の例として，文字列中に含まれる小文字の数を返す関数を示します．ここでは，DATA文により機械語プログラムを用意しておき，POKEにより書き込む方法により機械語プログラムを用意してます．

機械語ルーチンでは，まずAレジスタの値を調べて与えられた引数が文字型かどうかを調べています．ここでは，文字型でない場合はそのままBASICに戻ることにしています．

引数が文字型であった場合には，〔DE〕レジスタにより渡されたストリング・ディスクリプタの番地をもとに，文字列の長さと，実際に文字列の置かれている番地を求めます．

その後，ストリングを最初から1文字づつ調べていき，小文字であった場合にはANI命令によりD_5ビットを0にすることにより大文字に変換し，ストリングを変更します．このように，ユーザー関数は与えられた引数自身を変更するために，USR 0の呼び出し前後ではA$の内容が変化することになります．

```
100 '  USR function sample program
110 '
120 CLEAR ,&HE000  'reserve program area
130 MACHINE=&HE000
140 DEF USR0=MACHINE
150 '
160 FOR ADR=MACHINE TO MACHINE+30
170   READ BYTE:POKE ADR,BYTE 'poke ML program
180 NEXT ADR
190 '
200 INPUT A$
210   B$=USR0(A$)
220   PRINT B$
230 GOTO 200
240 END
250 '
260 '  machine language program data
270 '
280 DATA &hfe,&h03       :'cpi  3     ;argumet must string
290 DATA &hc0            :'rnz        ;if not,just ignore
300 DATA &heb            :'xhcg
310 DATA &h56            :'mov  d,m   ;d has length
320 DATA &h23            :'inx  h
330 DATA &h7e            :'mov  a,m
340 DATA &h23            :'inx  h
350 DATA &h66            :'mov  h,m
360 DATA &h6f            :'mov  l,a   ;hl=pointer to string
```

```
370 DATA &h14             :'inr  d      ;null string ?
380 '                loop:
390 DATA &h15             :'dcr  d      ;decrement length
400 DATA &hc8             :'rz          ;find string end ?
410 DATA &h7e             :'mov  a,m    ;load a char.
420 DATA &hfe,&h61        :'cpi  61h    ;char>'a' ?
430 DATA &hda,&h1b,&he0:'jc   notlow;No.
440 DATA &hfe,&h7b        :'cpi  7bh    ;char<='z' ?
450 DATA &hd2,&h1b,&he0:'jnc  notlow;No.
460 DATA &he6,&hdf        :'ani  0dfh   ;convert to upper
470 DATA &h77             :'mov  m,a    ;set it
480 '              notlow:
490 DATA &h23             :'inx  h
500 DATA &hc2,&h0b,&he0:'jmp  loop
```

D.4 CALL文

機械語ルーチンは，CALL 文を用いても呼び出すことができます．

引数を持たない CALL 文は機械語の CALL 命令と同等であり，指定された番地からのルーチンが呼び出されます．BASIC に戻るには，USR 関数と同じように RET 命令で戻ります．

CALL 文が引数を持っている場合には，レジスタ中に，引数で指定された変数の置かれている番地(VARPTR 関数で求められる番地と同一のもの)が準備された上で機械語ルーチンが呼び出されます(次節参照)．引数の番地情報を渡す方法は引数の数によって異なります．

(1) 引数が 3 個以下のとき

1 番目の引数の番地は〔HL〕レジスタに，

2 番目の引数の番地は〔DE〕レジスタに（もしあるならば)，

3 番目の引数の番地は〔BC〕レジスタに（もしあるならば)

準備されます．

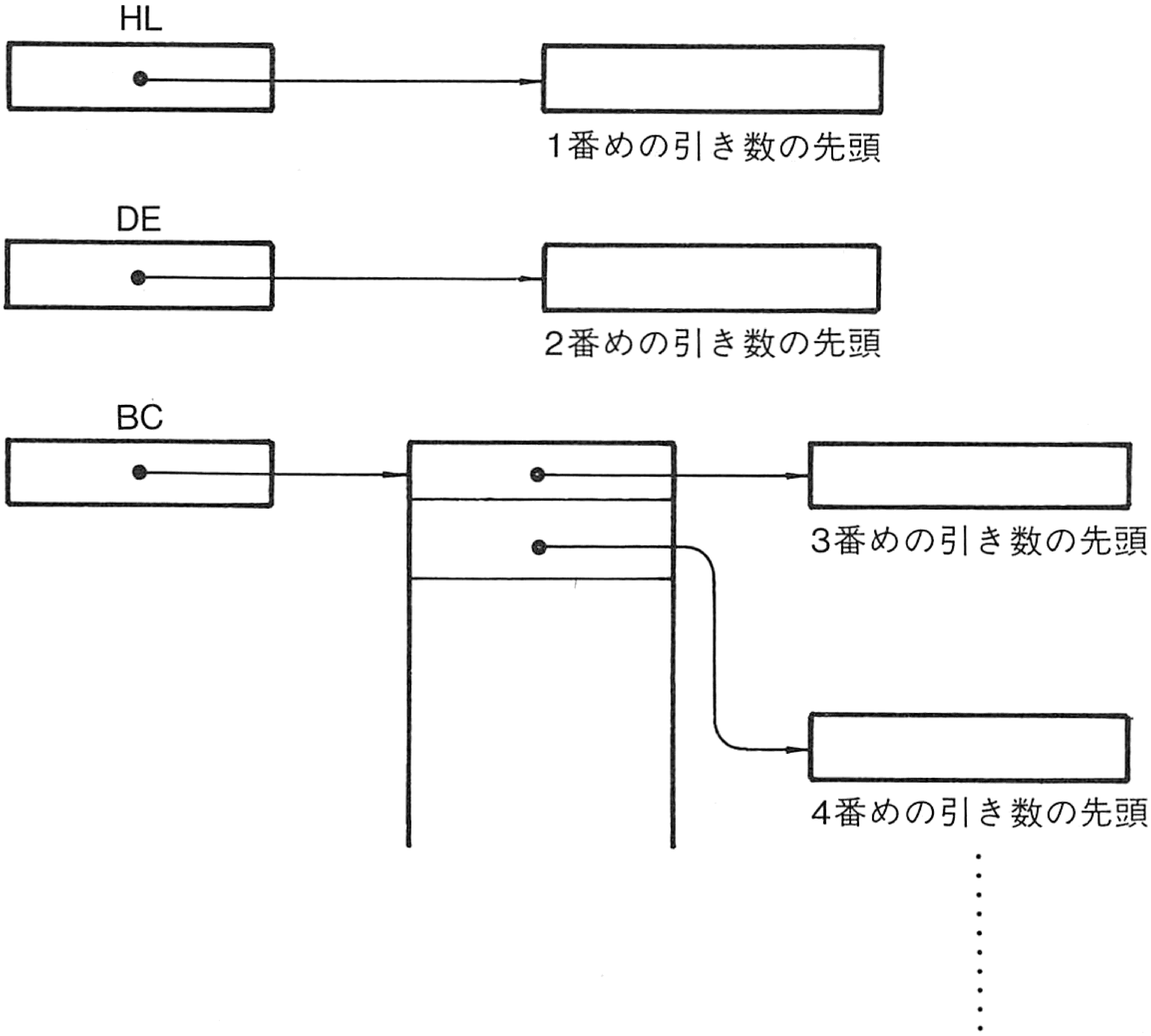
(2) 引数の数が 4 個以上のとき

1 番目の引数の番地は〔HL〕レジスタ，

2 番目の引数の番地は〔DE〕レジスタに

準備されます．

3 番目以降の引数については，それらの番地が順番に並べられた表が作成され，〔BC〕レジスタにその表の番地が準備されます．



CALL 文を用いる際には次の事項に注意してください.

- CALL 文が渡す引数の番地は，USR 関数の場合の FAC の番地ではなく，VARPTR 関数の値と同じ，変数値の置かれている番地です.

● CALL文では複数の引数を与えることができます．また与えられた引数の番地を介してBASICに値を返すことができます．

●引数が文字型の場合には準備される番地は，ストリング・ディスクリプタの番地です．

● CALL文は引数の個数や型については何ら情報を与えません．従って，これらを一致させることに注意しなければなりません．

サンプルプログラム

CALL文の例として，2つの数の最大値を与えるルーチンを示します．A%，B%により渡された2つの数のうちの大きい方の値をC%に代入します．

機械語ルーチンでは〔HL〕レジスタの持つ番地からA%，〔DE〕レジスタの持つ番地からB%の値を得，大きい方を〔BC〕レジスタの示すC%の番地に格納します．

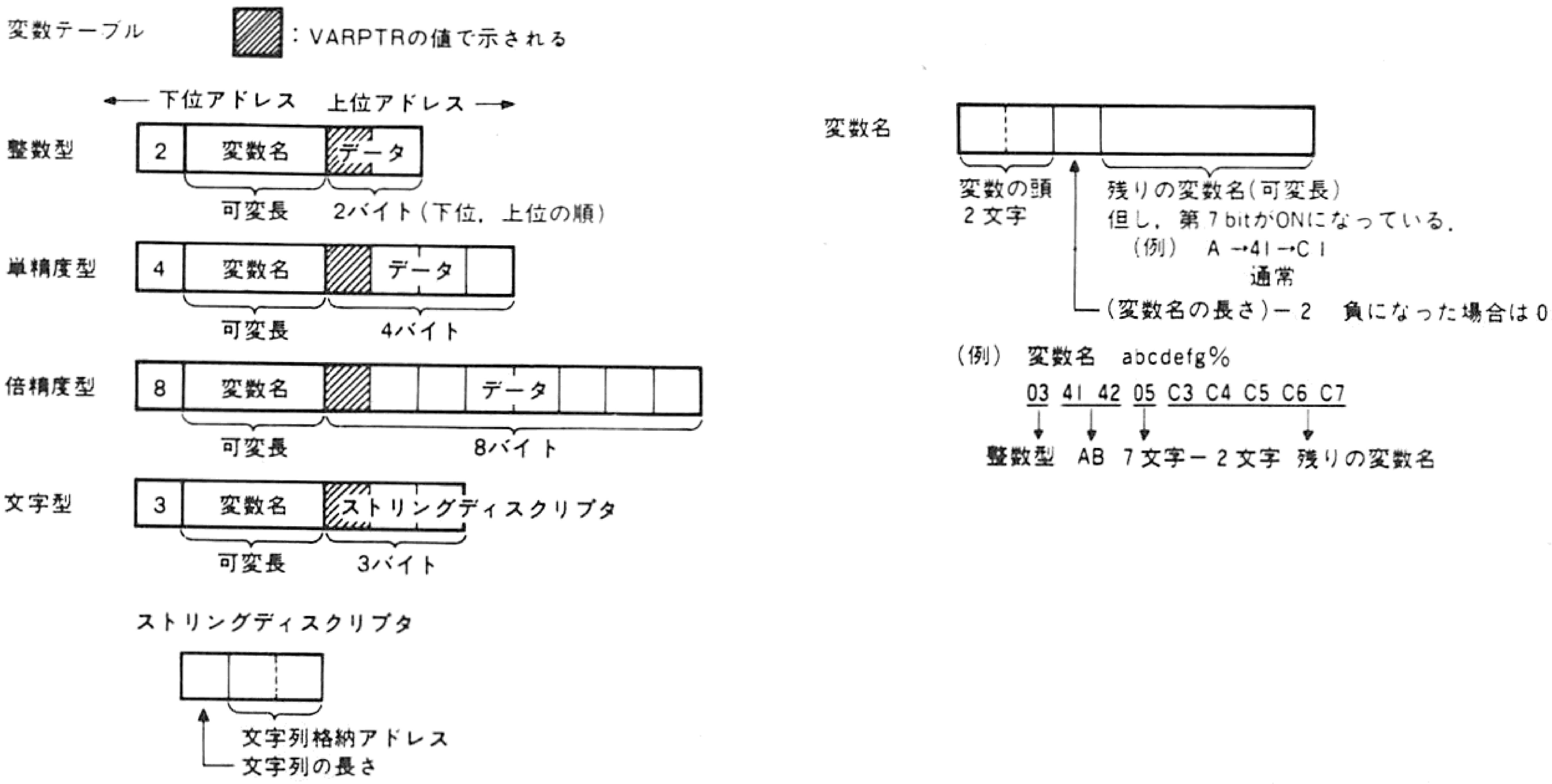
```
100 '  CALL statement sample program
110 '
120 CLEAR ,&HE000 'reserve program space
130 MAX=&HE000
140 '
150 FOR ADR=MAX TO MAX+35
160   READ BYTE:POKE ADR,BYTE 'poke ML language
170 NEXT ADR
180 '
190 C%=0
200 INPUT A%,B%
210   CALL MAX(A%,B%,C%)
220   PRINT C%
230 GOTO 200
240 END
250 '
260 '  machine language program data
270 '
280 '                    max:
290 DATA &hcd,&h1f,&he0:'call getval ;get a% value
300 DATA &heb           :'xchg
310 DATA &hcd,&h1f,&he0:'call getval ;get b% value
320 DATA &he5           :'push h      ;save b% value
330 DATA &hb7           :'ora  a      ;clear carry
340 DATA &h7b           :'mov  a,e    ;a%-b%
350 DATA &h95           :'sub  l
360 DATA &h7a           :'mov  a,d
370 DATA &h9c           :'sbb  h
380 DATA &h2e,&hff      :'mvi  l,0ffh
390 DATA &he2,&h13,&he0:'jpo  *+4    ;check v flag
400 DATA &h2c           :'inr  l
410 DATA &had           :'xra  l
420 DATA &he1           :'pop  h      ;assume b%<a%
430 DATA &hf2,&h19,&he0:'jp   *+4
```

```
440 DATA &heb               :'xchg              ;a%<=b%
450 DATA &h7d               :'mov   a,l         ;store into c%
460 DATA &h02               :'stax  b
470 DATA &h03               :'inx   b
480 DATA &h7c               :'mov   a,h
490 DATA &h02               :'stax  b
500 DATA &hc9               :'ret
510 '                 getval:
520 DATA &h7e               :'mov   a,m
530 DATA &h23               :'inx   h
540 DATA &h66               :'mov   h,m
550 DATA &h6f               :'mov   l,a
560 DATA &hc9               :'ret
```

D.5 VARPTR

VARPTR関数は指定された変数，バッファの置かれている番地を返します．

VARPTR関数の引数として変数名が与えられた時には，その変数の値の最下位バイトが置かれている番地を返します．上位バイトは下図に示すようにこの番地以降に入っています．また，文字変数名が指定された場合にはストリング・ディスクリプタの先頭番地が返されます．



変数として配列要素が指定された場合には，その要素の置かれている番地が返されます．

引数としてバッファ番号が与えられた場合には，そのバッファ領域の番地を返します．この領域の最初の9バイトはBASICの作業用の領域となっており，ファイルとの入出力に用いられる256バイトの領域は，VARPTR関数の値に9を加えた番地から始まります．

E 一 覧 表

エラーコード

注意：エラーメッセージの欄で，フルセンテンスで示されているものは，N88DISK-BASICまたは N-BASICのメッセージであり，カッコ内に示されている省略形のメッセージはN88-BASICのものです．エラーコードの欄は上段がN-BASIC，下段がN88-BASICのコードを表わしています．"—"で示されているのは，存在しないエラーです．

エラーメッセッセッージ	コード	意　　味
Bad allocation table	N 64 N88 69	ディスケット内部のFATが壊れている．
Bad drive number	N 65 N88 70	ドライブ指定が誤っている．システムにつながっていないドライブを指定した．
Bad File Data	N 25 —	ファイル上にあるデータの形式がまちがっている．
Bad file name (? NM Error)	N 62 N88 56	ファイル名の指定がまちがっている．
Bad file number (? BN Error)	N 52 N88 52	オープンしていないファイルや起動時に指定していないファイルを参照した．
Bad file mode	N 54 —	シーケンシャルでオープンしたファイルに対してランダムアクセスをしようとした．またはその逆．
Bad track / sector	N 66 N88 71	トラック，セクタ番号の指定が誤っている．(DSKO $, DSKI $)
Can't continue (? CN Error)	N 17 N88 17	CONT命令によって実行を再開することができない（ポインタの値が壊されている）．
Communications Buffer Overflow	N 27 —	周辺機器との入出力のためのバッファがオーバーフローした．（N88-BASICではLine buffer overflow）
Deleted record	N 67 N88 72	消去済みのレコードをアクセスしようとした．
Direct Statment in file (? DS Error)	N 63 N88 57	アスキー形式のファイルを読み込む際にダイレクトステートメントが存在した．
Disk already mounted	N 59 N88 67	mountされているディスクに対してmountを行った．
Disk BASIC Feature	N 26 —	ディスクが接続されていないとき，ディスクBASICの命令を実行した．

エラーメッセージ	コード	意　　味
Disk full	N 60 N88 68	ディスク上に書き込むスペースがないのに書き込もうとした．
Disk not mounted	N 56 N88 63	mount されていないディスクに対してアクセスした．
Disk I/O Error	N 57 N88 64	ディスクとの入出力中にエラーが発生した．致命的なエラーであり回復させることはできない．
Disk offline	N 73 N88 62	入出力の可能な状態でないディスクをアクセスしようとした．
Division by Zero (? /0 Error)	N 11 N88 11	0による割り算が実行されて，その結果の値がオーバーフローした．
Dupulicate Definition (? DD Error)	— N88 10	配列またはユーザー関数を二重定義しようとした．（N-BASIC では，Redimensioned array）
Dupulicate label (? DU Error)	— N88 31	同じラベル名が2つ以上存在している．
Feature not available (? NA Error)	— N88 33	利用不可能な機能を指定した．
FIELD overflow (? FO Error)	N 50 N88 50	FIELD 文において256バイト以上の大きさの領域を指定した．
File already exists	N 58 N88 65	NAME 文によって変更しようとしたファイル名が既に存在している。
File already open (? AO Error)	N 55 N88 54	既にオープンされているファイルに対して OPEN，KILL などを実行した．
File not found (?FF Error)	N 53 N88 53	LOAD，SAVE，KILL などで現在のディスクに存在しないファイルを指定した．
File not Open (? CF Error)	N 71 N88 60	オープンされていないファイルを参照しようとした．
File write protected	N 72 N88 61	書き込み禁止属性が付けられているファイルに書き込もうとした．
For without NEXT (? FN Error)	— N88 26	FOR～NEXT が正しく対応していない（FOR が多い）．
Illegal direct (? ID Error)	N 12 N88 12	ダイレクトモードで使用できない文を実行しようとした．

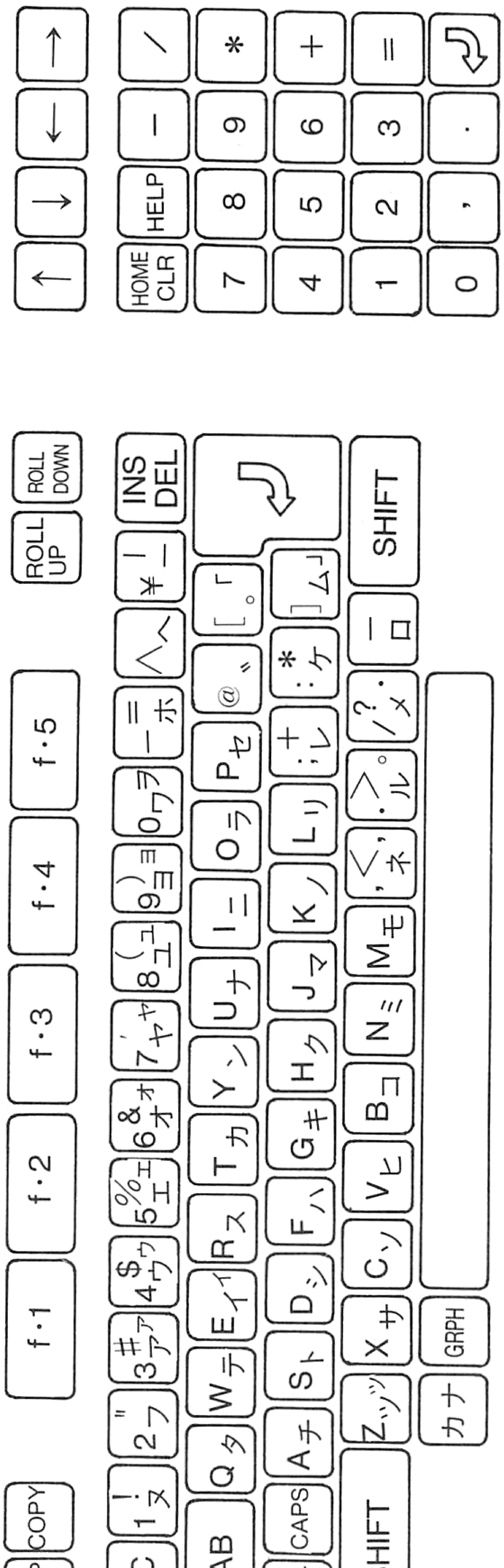
エラーメッセージ	コード	意　　　味
Illegal function call (? FC Error)	N 5 N88 5	ステートメントおよび関数において，機能の呼び方が誤っている．
Input past end (? EF Error)	N 61 N88 55	ファイル中の全てのデータを読み尽くした後に，さらに入力文が実行された．
Internal error (?IE Error)	N 51 N88 51	BASIC 内部にエラーが生じた．
Line buffer overflow (?BO Error)	N 23 N88 23	1行で入力できる文字の範囲を越えて入力が行われた．
Missing operand (? MO Error)	N 22 N88 22	ステートメント中必要なパラメータが指定されていない．
NEXT without FOR (? NF Error)	N 1 N88 1	FOR～NEXT が正しく対応していない． (NEXT が多い)
No RESUME (?NR Error)	N 19 N88 19	エラー処理ルーチンの中に RESUME 文がなく，プログラムの実行が継続できない．
Out of data (? OD Error)	N 4 N88 4	読むべきデータがないのに READ 文が実行された．
Out of memory (? OM Error)	N 7 N88 7	メモリ容量が足りなくなった（プログラムが大きすぎる，またはスタックを消費し尽した）．
Out of string space (? OS Error)	N 14 N88 14	文字列を格納するメモリ領域がなくなった．
Overflow (? OV Error)	N 6 N88 6	演算結果や入力された数値が，許される範囲を越えた．
Port not initialized	N 28 —	インターフェイス用の LSI の機能設定がなされていない．
Position not on Screen	N 24 —	指定したカーソル位置などが，画面の範囲外になっている．
Redimensioned array	N 10 —	配列またはユーザー関数を二重定義しようとした（N88-BASIC では Dupulicate Definition）．
Rename across disks	N 68 N88 73	NAME 文において，異なるドライブ間でリネームしようとした．
RESUME without error (? RW Error)	N 20 N88 20	エラー処理ルーチンに制御を移さなかったのに RESUME 文がある．

エラーメッセージ	コード	意　　味
RETURN without GOSUB (? RG Error)	N 3 N88 3	GOSUB～RETURNが正しく対応していない （RETURNが多い）.
Sequential after PUT (? SP Error)	N 69 N88 58	PUT 文実行後，シーケンシャルファイルをアクセスしようとした.
Sequential I/O Only (?SQ Error)	N 70 N88 59	シーケンシャル入出力以外は行ってはならない.
String formula too complex (? ST Error)	N 16 N88 16	文字式が複雑すぎる. （カッコのネスティングが深すぎるなど）
String too long (? LS Error)	N 15 N88 15	1つの文字変数内の文字数が255文字を越えている.
Subscript out of range (? BS Error) (Bad Subscript)	N 9 N88 9	配列変数の添字の値が規定の範囲から外れている.
Syntax error (? SN Error)	N 2 N88 2	文の記述が文法通りになっていない.
Tape read error (? TP Error)	N 29 N88 27	テープからの読み込み時にエラーが発生した.
Type mismatch (? TM Error)	N 13 N88 13	式の左辺・右辺，関数の引数などにおいて変数の型が一致していない.
Undefined label (? UN Error)	— N88 32	定義されていないラベル名で参照した.
Undefined line number (? UL Error)	N 8 N88 8	飛び先として必要とされる行番号（GOTO, GOSUBなどの飛び先）が存在しない.
Undefined user function (? UF Error)	N 18 N88 18	DEFFN 文によって定義されていないユーザー関数を引用しようとした.
Unprintable error (? UE Error)	N 21 N88 21	メッセージの定義されていないエラー.
WEND without WHILE (? WE Error)	— N88 30	WHILE～WENDが正しく対応していない. （WENDが多い）.
WHILE without WEND (? WH Error)	— N88 29	WHILE～WENDが正しく対応していない. （WHILEが多い）.

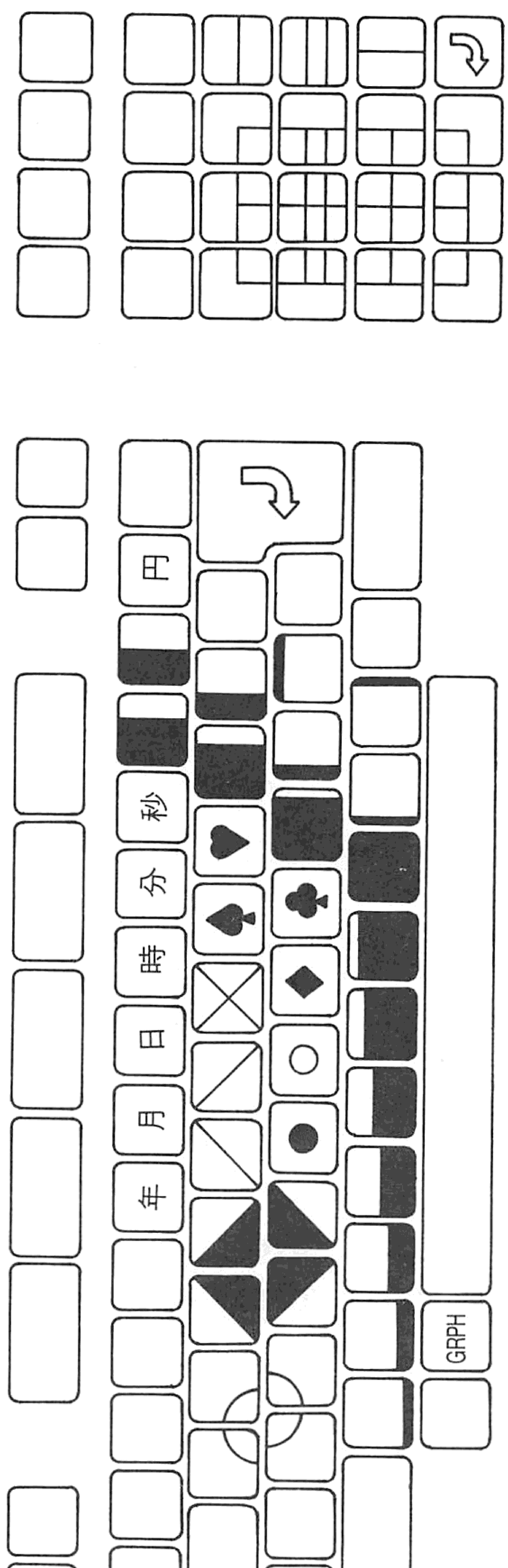
コントロールコード表

16進	10進	対応するキー	N88-BASIC	N-BASIC
01	1	CTRL + A	ヘルプキーと同じ	
02	2	CTRL + B		1つ前のワードへ戻る
03	3	CTRL + C	実行の中断	(N88-BASICと同じ)
04	4	CTRL + D	カーソル位置から1ワードを削除	
05	5	CTRL + E	カーソル位置から後を消す	(N88-BASICと同じ)
06	6	CTRL + F	1つ先のワードへ進む	
07	7	CTRL + G	スピーカを鳴らす	
08	8	CTRL + H	カーソル位置の左側の文字を削除する	(N88-BASICと同じ)
09	9	CTRL + I	水平タブ(8文字毎)	(N88-BASICと同じ)
0A	10	CTRL + J	ラインフィード，インサートモードで2行に分割	行を2つに分割
0B	11	CTRL + K	ホームポジション	(N88-BASICと同じ)
0C	12	CTRL + L	テキスト画面クリア	(N88-BASICと同じ)
0D	13	CTRL + M	キャリッジリターン	(N88-BASICと同じ)
0E	14	CTRL + N		1つ先のワードへ進む
0F	15	CTRL + O	画面の表示を無効にする	
12	18	CTRL + R	インサートモードにする	カーソル位置から右側を1文字分右へずらす
13	19	CTRL + S	実行を一時停止する	
15	21	CTRL + U	1行キャンセル	
18	24	CTRL + X	カーソルを行の最後に移す	
1B	27	ESC		実行を一時停止する
1C	28	→	カーソルを右へ移動	(N88-BASICと同じ)
1D	29	←	カーソルを左へ移動	(N88-BASICと同じ)
1E	30	↑	カーソルを上へ移動	(N88-BASICと同じ)
1F	31	↓	カーソルを下へ移動	(N88-BASICと同じ)

キーボード

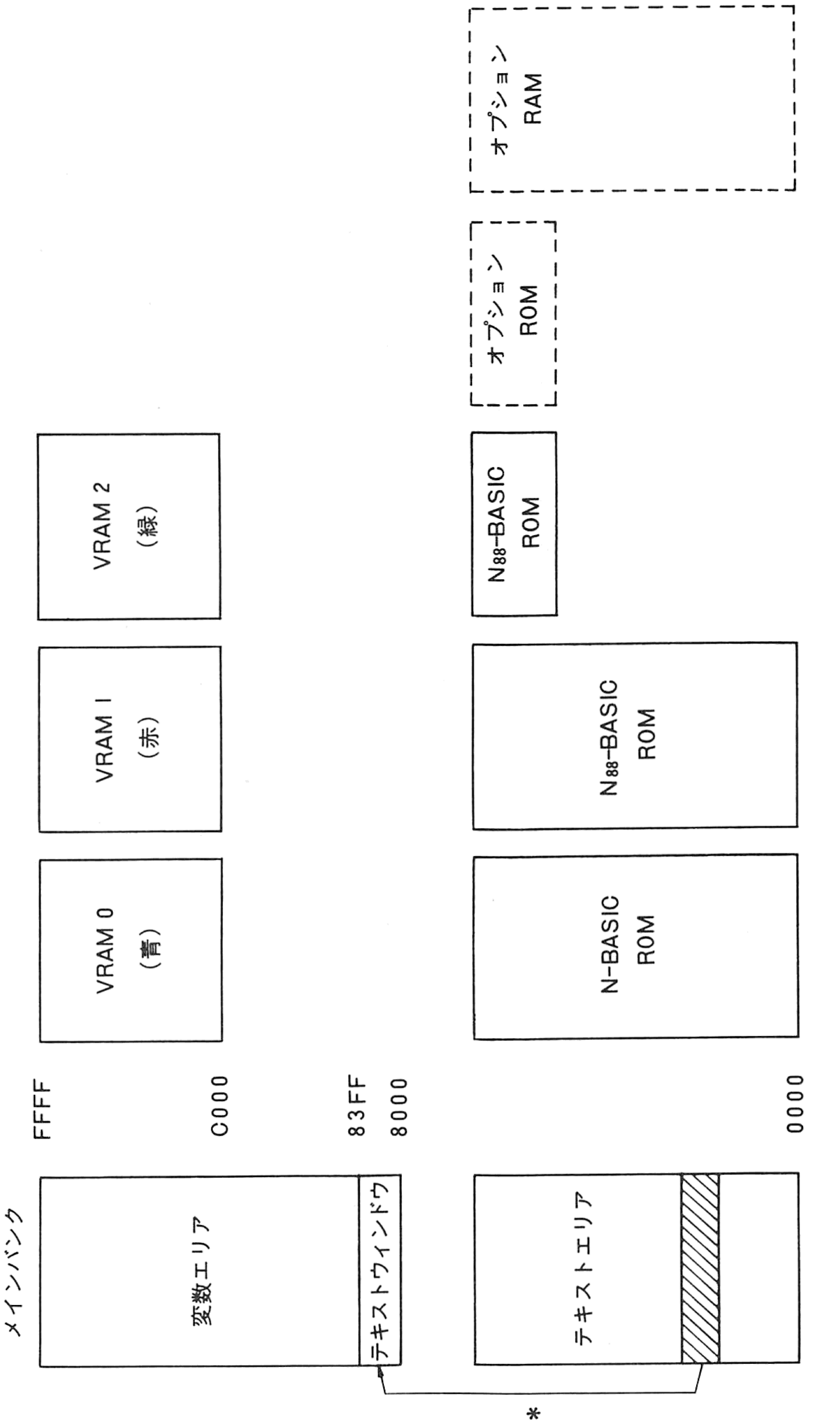


キーの配列



グラフィックシンボルキーの配列

メモリマップ



＊：N88-BASICモードでは，テキストエリアの1Kバイトが変数エリアのテキストウィンドウに転送されて処理される.

キャラクタ

上位4ビット→

下位4ビット↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D_E		0	@	P		p				ー	タ	ミ		
1	S_H	D_1	!	1	A	Q	a	q			。	ア	チ	ム		円
2	S_X	D_2	"	2	B	R	b	r			「	イ	ツ	メ		年
3	E_X	D_3	#	3	C	S	c	s			」	ウ	テ	モ		月
4	E_T	D_4	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	E_Q	N_K	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	A_K	S_N	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	B_L	E_B	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	B_S	C_N	(	8	H	X	h	x			ィ	ク	ネ	リ	♠	
9	H_T	E_M	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	♥	
A	L_F	S_B	*	:	J	Z	j	z			ェ	コ	ハ	レ	♦	
B	H_M	E_C	+	;	K	[	k	{			ォ	サ	ヒ	ロ	♣	
C	C_L	→	,	<	L	¥	l	¦			ャ	シ	フ	ワ	●	
D	C_R	←	−	=	M	]	m	}			ュ	ス	ヘ	ン	○	
E	S_O	↑	.	>	N		n	~			ョ	セ	ホ	゛		
F	S_I	↓	/	?	O	_	o				ッ	ソ	マ	゜		

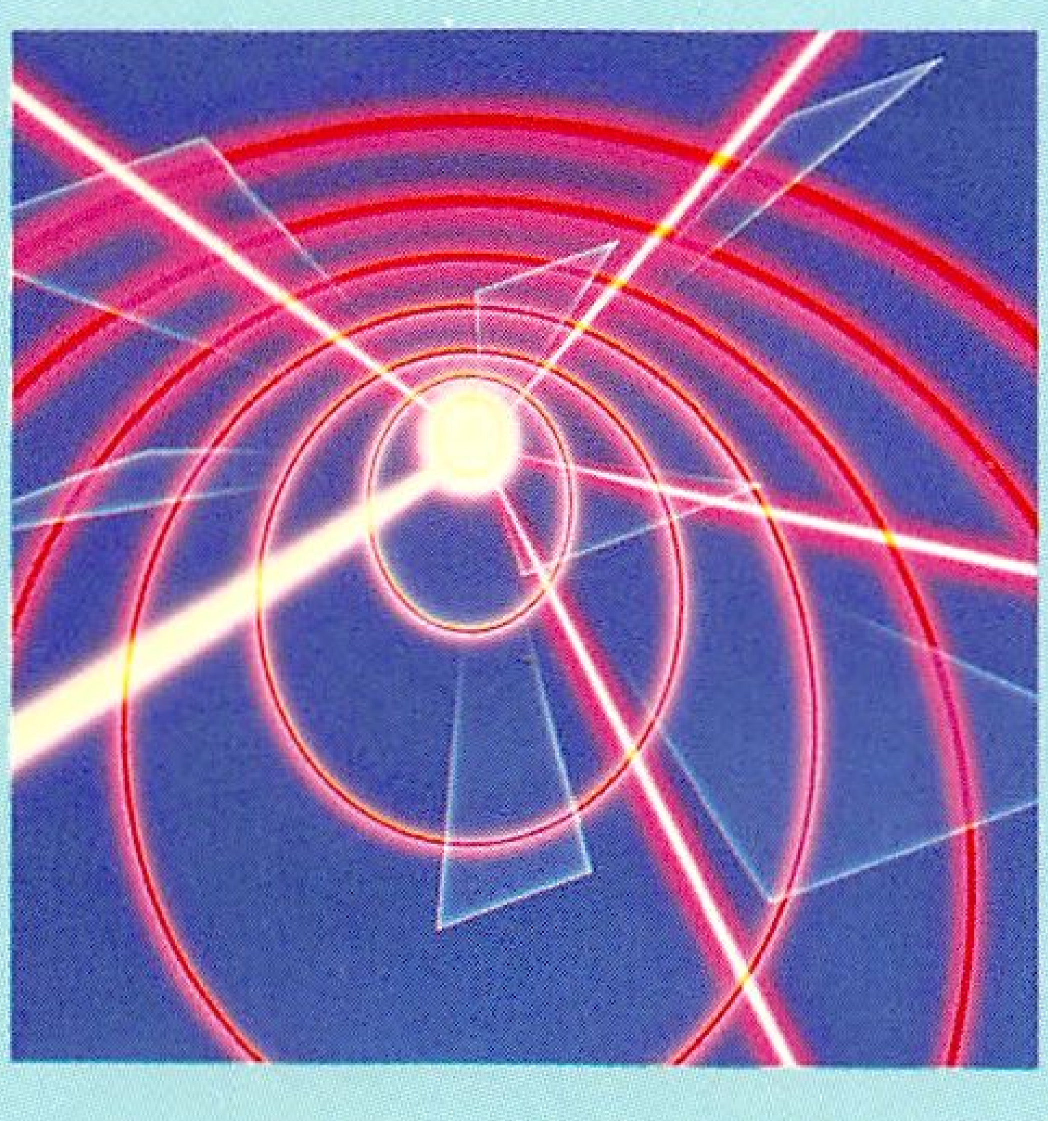
ファイルディスクリプタ

デバイス名	機械名称	入力	出力	備　　考
LPT1： またはLPT:	プリンタ	×	○	
SCRN：	スクリーン	×	○	N88DISK-BASICで使用可
COM1： またはCOM:	RS-232Cポート 1	○	○	
CAS1： またはCAS:	カセットテープ 1	○	○	N88-BASICでのデフォルト，転送速度1200ボー
CAS2：	〃 2	○	○	転送速度600ボー
1：	フロッピーディスク 1	○	○	N88DISK-BASICでのデフォルト
2：	〃 2	○	○	
3：	〃 3	○	○	
4：	〃 4	○	○	
5：	〃 5	○	○	
6：	〃 6	○	○	
7：	〃 7	○	○	
8：	〃 8	○	○	
KYBD：	キーボード	○	×	N88DISK-BASICで使用可

F　誘導関数

N88(N)－BASICが“組み込み”関数として用意していない三角関数のうち，いくつかは，用意された関数を使って作り出すことができます．以下の数式を参考にしてください(誤差の範囲に注意が必要です)．

目的とする関数	組み込み関数からの誘導式
セカント	SEC(X)＝1/COS(X)
コセカント	CSC(X)＝1/SIN(X)
コタンジェント	COT(X)＝1/TAN(X)
アークサイン	ARCSIN(X)＝ATN(X/SQR(－X*X＋1))
アークコサイン	ARCCOS(X)＝－ATN(X/SQR(－X*X＋1))＋1.5708
アークセカント	ARCSEC(X)＝ATN(SQR(X*X－1))＋(SGN(X)－1)*1.5708
アークコセカント	ARCCSC(X)＝ATN(1/SQR(X*X－1))＋(SGN(X)－1)*1.5708
アークコタンジェント	ARCCOT(X)＝－ATN(X)＋1.5708
ハイパーボリック・サイン	SINH(X)＝(EXP(X)－EXP(－X))/2
ハイパーボリック・コサイン	COSH(X)＝(EXP(X)＋EXP(－X))/2
ハイパーボリック・タンジェント	TANH(X)＝－EXP(－X)/(EXP(X)＋EXP(－X))*2＋1
ハイパーボリック・セカント	SECH(X)＝2/(EXP(X)＋EXP(－X))
ハイパーボリック・コセカント	CSCH(X)＝2/(EXP(X)－EXP(－X))
ハイパーボリック・コタンジェント	COTH(X)＝EXP(－X)/(EXP(X)－EXP(－X))*2＋1
ハイパーボリック・アークサイン	ARCSINH(X)＝LOG(X＋SQR(X*X＋1))
ハイパーボリック・アークコサイン	ARCCOSH(X)＝LOG(X＋SQR(X*X－1))
ハイパーボリック・アークタンジェント	ARCTANH(X)＝LOG((1＋X)/(1－X))/2
ハイパーボリック・アークセカント	ARCSECH(X)＝LOG((SQR(－X*X＋1)＋1)/X)
ハイパーボリック・アークコセカント	ARCCSCH(X)＝LOG((SGN(X)*SQR(X*X＋1)＋1)/X)
ハイパーボリック・アークコタンジェント	ARCCOTH(X)＝LOG((X＋1)/(X－1))/2
常用対数	LOG_{10}(X)＝LOG(X)/LOG(10)



NEC 日本電気株式会社・日本電気ホームエレクトロニクス株式会社

78111833